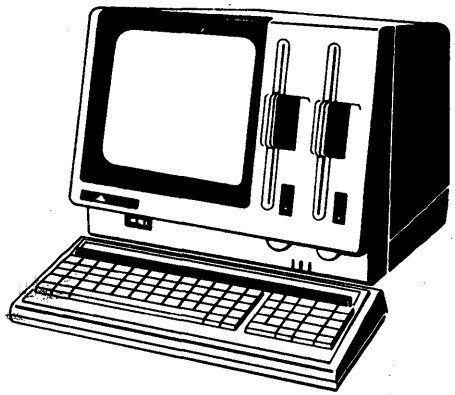
Advanced
Personal Computer



dBase II User's Guide

NEC
NEC Information Systems, Inc.
819-000100-8001
2-83

Contents

Contents (cont'd)

Contents (cont'd)

Contents (cont'd)

Contents (cont'd)

Contents (cont'd)

Contents (cont'd)

Figures

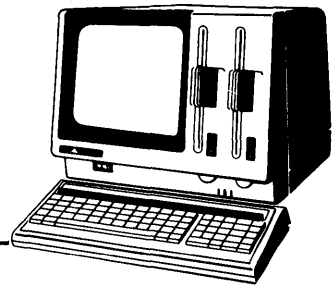
Tables

Preface

The dBASE II system for the NEC Advanced Personal Computer (APC) is distributed with a diskette containing the dBASE II programs and several sample applications and data bases. The *dBASE II User's Guide* for the APC is a tutorial guide and reference manual for use with dBASE II.

The first part of this guide is a tutorial. It explains the basic dBASE II commands and leads you through the processes of defining and manipulating data bases. To run the sample application described in Appendix H of this manual, you must create the data base described in the tutorial.

The appendices at the end of the manual are the reference materials. Appendix A through Appendix G summarize the concepts, commands, functions, and procedures presented in the tutorial section of the manual. Appendix H describes one sample application. Appendix I defines the function key assignments. Appendix J contains listings of programs created by dBASE II users like you. It will give you an idea of the kinds of command files you can create using this system.



Chapter 1

Introduction to dBASE II

dBASE II is a data base management tool that allows easy manipulation of small and medium sized data bases using English-like commands. With dBASE II and the NEC Advanced Personal Computer (APC) you can:

- create complete data base systems;
- easily add, delete, edit, display, and print data from your data base;
- maintain data files with a minimum of data duplication;
- generate reports from one or more data bases, automatically producing subtotals, totals, headings, and subheadings.

This guide is designed to be both a tutorial and a reference manual. It assumes that you have never used a data base management system or written a computer program. The guide takes you through the processes of designing, creating, and using a data base in tutorial format. It also presents complete descriptions of the dBASE II commands and examples for practice.

It's easy to learn to use dBASE II on the APC. You can understand how to use the basic features in one day. Advanced functions, such as programming, are built on the simple commands and are easy to learn once you know the basics.

USING THIS GUIDE

The best way to learn how dBASE II works is to use it. Take the time to sit down at your APC with this guide and try the commands. Use the examples presented and make up your own. Don't worry about making mistakes. It's hard to ruin a data base with dBASE II. Anything you can do you can probably undo just as easily. So experiment with the commands.

Once you are familiar with dBASE II, you can use this manual for reference. It includes alphabetical summaries of commands and functions, a table of edit features, and other useful appendices. You will also find the Reference Card helpful as a quick reminder of command formats and options.

THE EQUIPMENT

This section describes features of the APC keyboard, screen, diskette drives, diskettes, and printer that apply specifically to their operation with dBASE II.

The Keyboard

The APC keyboard is similar to a typewriter keyboard. The *alphanumeric keyboard* includes the typical typewriter keys. On the right is a smaller *numeric keypad* that contains the numbers 0-9, arithmetic symbols, a set of *cursor movement keys,* and a few keys that have special uses in dBASE II and are defined later in this guide. Across the top of the keyboard are *program function keys.* The first key is labelled FNC. The next 22 keys are unlabelled, but are referred to as **PF1** through **PF22**. You can use **PF1** through **PF12**, the first twelve function keys, to simplify your work. Take out the template that is packaged with this guide. Insert it in the slot above the function keys. The template labels twelve function keys for dBASE II.

To use the function keys, simply press the key below the command named on the template instead of typing the command at the dBASE II prompt. The system displays the command on the screen in either its full or abbreviated form, as if you had typed it. If the command accepts modifiers, you can then type them after the command. See Appendix I for a complete list of function key assignments and the corresponding screen displays.

NOTE

In this guide, key names and the information you are asked to type appear in **bold.** For example, program function key 2 is **PF2,** the escape key is **ESC,** the letter A is **A,** and so forth.

The Screen

dBASE II screen displays are monochromatic, even if you are using an APC color terminal.

The Diskette Drives

dBASE II can be used with both single-drive and dual-drive models of the APC. If you have a dual-drive model, you can use both disk drives with dBASE II. You can keep the dBASE II programs on the diskette in Drive A and the data on a separate diskette in Drive B. With single-drive and dual-drive models, you can store both the dBASE II programs and your data base files on the same diskette.

The Diskettes

A *diskette* is a thin circle of recording material inside a square plastic covering. Diskettes are light and portable, yet each diskette holds more information than many pieces of paper.

Since all your data are stored on diskettes, they are very valuable even though each diskette costs only a few dollars. Diskettes require special attention to keep them in good condition. To protect your work, follow these suggestions.

- Always make copies (called backups) of important data base files on a separate diskette. Every diskette eventually wears out. If you don't have copies of your data base files, they can be lost.
- Never touch the diskette's recording surface. Handle a diskette by the plastic covering. Be careful not to touch the exposed parts of the diskette surface.
- Always return the diskette to its protective jacket when you aren't using it. Most diskette drives read and write on the back side of the diskette, so don't lay a diskette down face up without its jacket.
- Never write on the diskette. To mark it, write on a gummed label, then stick it on the diskette.
- Do not expose the diskette to extreme heat (over 90 degrees), direct sunlight, or magnetic fields. Scissors, paper clips, and other metal items may be magnetic. Be careful - they can erase or change information on the diskette.

The Printer

You can use dBASE II on the APC without a printer if you do not require hard copy versions of reports, file lists, dBASE II programs, and other output from the system.

CONVENTIONS USED IN THIS MANUAL

This section describes the typographic conventions, special symbols, and instruction conventions used in this guide. Read this carefully before using dBASE II with the APC.

Typographic Conventions

Lowercase letters in the text and screen displays indicate material that you type. You can use either uppercase or lowercase, however, when you actually type your entries.

Uppercase letters in the text identify dBASE II commands and keywords. In the screen displays, uppercase is used for the dBASE II system's prompts and responses.

Keys are named as they appear on the keyboard and are printed in the text in **bold.** **RETURN** means the key labelled "RETURN", **ESC** identifies the key labelled "ESC", **P** means the letter "P", and so on. Do not type the word. Just press the indicated key.

Special Symbols

The following *special symbols* are used in dBASE II commands.

[...] Square brackets indicate parts of commands that are optional. Do not type the symbols.

<...> Pointed brackets enclose portions of commands that are to be filled in with real information. Do not type either the symbols or the lowercase words and abbreviations they enclose. Replace them entirely with your information. For example, <file> should be replaced with the name of a file.

(...) Parentheses are required in some commands. When they appear in the command format, they must be used in the command you enter.

Terminology

Two other conventions are used in this guide to describe how to key data into the system.

Press means to press the named key, as in "Press **RETURN**" and "Press **PRINT**".

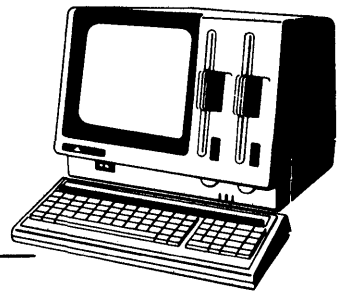
Enter identifies the two-step process of typing information and pressing **RETURN**. For example, you will find instructions like the following.

```
At the dot prompt, enter:
  create employee
```

This means that you are to type the two words "create employee" in either uppercase or lowercase and then press **RETURN**.

NOTE

The **RETURN** key on the alphanumeric keyboard and the **ENTER** key on the numeric keypad perform the same function and may be used interchangably. It is most convenient to press **ENTER** following numeric input and **RETURN** following alphanumeric input.



Chapter 2

Getting Started

BEFORE USING THE dBASE II DISKETTE, CAREFULLY FOLLOW THE PROCEDURES OUTLINED IN THIS CHAPTER TO MAKE A WORKING COPY OF THE dBASE II DISKETTE. To do this, you need the CP/M-86 operating system diskette, the dBASE II diskette, and a blank diskette.

This chapter provides step-by-step instructions for all the procedures required to create a working copy of dBASE II. The *working copy* refers to the diskette that is used regularly to perform dBASE II functions. The *original copy* refers to the distribution diskette you purchased. It should be kept in a safe place and used only to make new working copies. It is a good procedure to make a *backup copy* of data base files regularly, especially after making changes to them. Keep the backup copies on a diskette used only for this purpose. This insures that you always have a diskette with a copy of your data.

To make a working copy of dBASE II, you must prepare a new diskette, copy the original dBASE II diskette onto the new diskette, and put the CP/M-86 operating system, the system loader, and the autostart program on the new diskette.

FORMATTING DISKETTES

The original dBASE II diskette is ready to be used on the NEC Advanced Personal Computer (APC). However, to make a working copy of dBASE II, you must first prepare (or "format") another diskette.

A program called FORMAT is supplied with the CP/M-86 operating system for this purpose. This program analyzes diskettes for any defective areas and prepares the diskette for accepting CP/M-86 and dBASE II files.

To format a diskette, follow this procedure:

1. Turn on the APC. Insert the CP/M-86 operating system diskette into Drive A. Always insert a diskette so that the label faces toward the screen. The label should go in last. Close the door to the diskette drive.

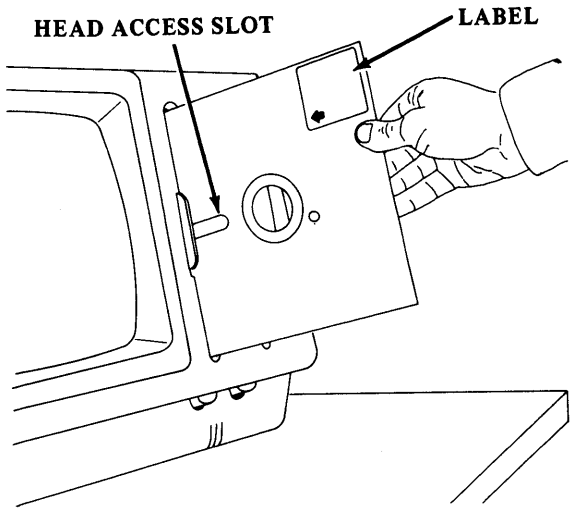


Figure 2-1 Inserting a Diskette

CAUTION

Never insert any diskette before the APC is powered on and never power down until all diskettes are removed.

Never insert a diskette into a drive when the red light next to the drive is lit. If this happens, remove the diskette and turn off the APC. Turn the system on again, insert the diskette, and close the door.

You will hear the diskette spinning in the drive. The following prompt is displayed.

```
A>
```

2. Type **FORMAT**. You may use either uppercase or lowercase characters.
3. Press **RETURN**. When the FORMAT program is loaded, the following prompt appears.

```
*** VOLUME INITIALIZER V1.1 (CP/M-86) ***
INITIALIZE DEVICE [A,B,C,D] :
```

4. Type the letter of the drive into which you are inserting the diskette to be formatted.

   - If you have a dual-drive APC, insert the diskette to be formatted into Drive B and type **B** at the prompt.
   - If you have a single-drive APC, remove the CP/M-86 operating system diskette by squeezing the diskette drive door latch. (The spring-lock mechanism releases the diskette as the drive door opens.) Then, insert the diskette to be formatted. Type **A** at the prompt.

NOTE

Diskette Drives C and D are not currently used with the APC.

5. Press **RETURN**. A prompt similar to the following prompt appears.

```
MEDIA TYPE FD-2D
READY? [R:READY A:ABORT] :
```

The abbreviation following "MEDIA TYPE" identifies the type of diskette you inserted. FD-1D indicates a single-sided, single-density diskette. FD-2D indicates a double-sided, double-density diskette.

The "READY? [R:READY A:ABORT]" prompt allows you to review your actions before proceeding.

WARNING

The FORMAT program destroys all existing information on the diskette. Before proceeding, review the entries made and make sure that the diskette you are formatting is in the proper drive.

6. If you made a mistake or you want to change your entry and return to the "INITIALIZE DEVICE [A,B,C,D]" prompt, enter **A** and return to Step 4.

   If you are ready to continue the formatting process, enter **R**. As formatting occurs, the number of the cylinder being formatted appears in brackets as shown below.

   ```
   REFORMATING CYLINDER NO. [00]
   ```

   If a damaged cylinder is detected during formatting, the following prompt displays the number of the damaged cylinder.

   ```
   * BAD CYLINDER NO. [47]
   [R:RETRY A:ABORT] :
   ```

   Enter **R** to try to format the cylinder again. If the damaged cylinder cannot be formatted, enter **A** to abort the procedure. The program returns to the "INITIALIZE DEVICE [A,B,C,D]" prompt. Remove the diskette and label it damaged. Insert another diskette in the appropriate drive and try again.

   When all cylinders on a diskette are successfully formatted, the following prompt appears.

   ```
   * NORMAL END
   [A:AGAIN E:END]      :
   ```

7. Remove the formatted diskette by squeezing the diskette drive door latch and then releasing it. The spring-lock mechanism releases the diskette as the drive door opens.

   To repeat the process, enter **A** and go back to Step 4.

   To end the program, enter **E**. The CP/M-86 "A>" prompt reappears.

MAKING A WORKING COPY

A program called COPYDISK is supplied with the CP/M-86 operating system diskette for making copies of all program and data files on a diskette. It physically copies all information from one diskette to another. Use this program to make a working copy of dBASE II. Then store the original diskette in a safe place and use it only for making new working copies. Also use COPYDISK periodically to make backup copies of often-used diskettes.

To use COPYDISK, follow this procedure:

1. Insert the CP/M-86 operating system diskette into Drive A. The "A>" prompt is displayed.
2. Type **COPYDISK** and press **RETURN**. When the program is loaded, the following prompt appears.

   Full Disk Copy/Verify Utility V1.1
   Copy or Verify or Copy & Verify (C,V,B)?
3. Select one of the following three options:

   C to copy the contents of one diskette to another;
   V to verify the contents of one diskette with another;
   B to both copy and verify a diskette.

   When copying diskettes, it is advisable to also verify the information. To copy and verify, type **B** in response to the prompt.

   NOTE

   With the COPYDISK program, you do not have to press **RETURN** after typing responses.

   The following prompt appears.

   Source Disk Drive (A-D)?
4. Before responding to this prompt, remove the CP/M-86 operating system diskette from Drive A and insert the original dBASE II diskette into Drive A.
5. Type **A** to indicate that you are copying from that drive. The following prompt appears.

   Destination Disk Drive (A-D)?
6. Identify the diskette drive you are copying to.

   - If you have a dual-drive APC, insert a formatted diskette into Drive B. Type **B** at the prompt to indicate you are copying to that drive.
   - If you have a single-drive APC, type **A** in response to the prompt.

The following prompt names both the source and destination drives to allow you to review your entries.

```
Copying Disk A: To Disk B:
Is This What You Want To Do (Y/N)?
```

WARNING

The COPYDISK program replaces all existing information on the destination diskette. Before proceeding, review the entries and make sure the source diskette is in the proper drive.

7. If you made a mistake or want to change the entries, type **N** in response to the prompt and return to Step 3.

   If you are ready to continue, type **Y**. What happens next depends on the type of APC you have.

   - If you have a dual-drive APC, the following display appears as each cylinder of the diskette in Drive A is copied to the diskette in Drive B.

     ```
     Copy Started
      Reading Cylinder     [00]
      Writing Cylinder     [00]
     ```

   - If you have a single-drive APC, the COPYDISK program reads a portion of the source diskette, then prompts you to remove the source diskette and insert the destination diskette. The program then writes to the destination diskette and prompts you to reinsert the source diskette.

     This procedure is repeated until the entire source diskette is copied. The system displays the following series of messages.

     ```
     Copy Started
      Reading Cylinder     [00]
     Insert "DESTINATION" Diskette In Drive A:
     Strike Any Key When Ready
      Writing Cylinder     [00]
     Insert "SOURCE" Diskette In Drive A:
     Strike Any Key When Ready
     ```

When copying is completed, the COPYDISK program verifies the contents of each diskette to insure that an exact duplicate was made. This procedure also depends on the type of APC you have.

- If you have a dual-drive APC, the following display appears as the corresponding cylinders on the diskettes are verified.

```
Verify Started
 Reading Cylinder    [00]
 Writing Cylinder    [00]
```

- If you have a single-drive APC, you must insert and remove the source and destination diskettes when prompted as the information on each diskette is verified. The procedure, represented by the following series of messages, is repeated until verification is completed.

```
Verify Started
 Reading Cylinder    [00]
Insert "DESTINATION" Diskette In Drive A:
Strike Any Key When Ready
 Verifying Cylinder  [00]
Insert "SOURCE" Diskette In Drive A:
Strike Any Key When Ready
```

When verification is completed, the following prompt appears.

```
Copy/Verify Another Disk (Y,N) ?
```

8. To repeat the process with another disk, type **Y** and go back to Step 3.

   To end the program and return to the CP/M-86 "A>" prompt, type **N**.

PUTTING CP/M-86 AND SUBMIT ON THE dBASE II DISKETTE

Before you can use the working copy of the dBASE II program diskette you created with COPYDISK, you must put two programs on the diskette. These two programs are the CP/M-86 operating system and a program that automatically starts the dBASE II program.

The Peripheral Interchange Program (PIP) is used for this. Unlike the COPYDISK program, which duplicates entire diskettes, PIP copies individual programs and files from one diskette to another. The files you must "PIP" from the CP/M-86 operating system diskette onto the working copy of dBASE II are called CPM.SYS and SUBMIT.CMD.

The procedure to copy files using PIP is different for single-drive and dual-drive APCs. Use the appropriate procedure for your APC.

Using PIP With A Dual-Drive APC

To use PIP with the dual-drive model of the APC, follow this procedure:

1. Insert the CP/M-86 operating system diskette into Drive A. The "A>" prompt is displayed.
2. Insert the working copy of dBASE II into Drive B.
3. Enter the following command.

   PIP B:=CPM.SYS[v]

   The PIP program copies the operating system from Drive A to Drive B, then returns to the "A>" prompt.
4. Enter the following command.

   PIP B:=SUBMIT. CMD [v]

   The PIP program copies the SUBMIT program from Drive A to Drive B, then returns to the "A>" prompt.

Using PIP With a Single-Drive APC

To use PIP with the single-drive model of the APC, follow this procedure:

1. Insert the CP/M-86 operating system diskette into the disk drive. The "A>" prompt is displayed.
2. Enter the following command.

   PIP1 CPM.SYS=CPM.SYS

   The display screen clears and the system displays the following prompt.

   ```
   INSERT SOURCE DISKETTE INTO DRIVE A
   PRESS RETURN TO CONTINUE:
   ```
3. Press **RETURN**.

   As the system reads CPM.SYS from the CP/M-86 operating system diskette, it displays the following message at the top of the screen.

   ```
   TRANSFERRING
   -CPM     .SYS-
   ```

   Then, the following prompt appears.

   ```
   INSERT DESTINATION DISKETTE INTO DRIVE A
   PRESS RETURN TO CONTINUE:
   ```

4. Remove the CP/M-86 operating system diskette from the disk drive and insert the working copy of dBASE II into the disk drive. Then, press **RETURN**.

   PIP1 writes CPM.SYS onto the working copy diskette. When the CPM.SYS file is copied, the PIP1 program again displays the following prompt.

   ```
   INSERT SOURCE DISKETTE INTO DRIVE A
   PRESS RETURN TO CONTINUE:
   ```

5. Insert the CP/M-86 operating system diskette into the disk drive again and press **RETURN**. The following message is displayed.

   ```
   TRANSFER COMPLETE
   -CPM      .SYS-
   ```

   The system then displays the following prompt.

   ```
   A (AGAIN) OR E (END)?
   ```

6. Press **SHIFT E.** (This prompt must be answered in uppercase.) The «A>« prompt reappears.

7. Enter the following command.

   PIP1 SUBMIT.CMD = SUBMIT.CMD

   The display screen clears and the system displays the following prompt.

   ```
   INSERT SOURCE DISKETTE INTO DRIVE A
   PRESS RETURN TO CONTINUE:
   ```

8. Press **RETURN**.

   As the system reads SUBMIT.CMD from the CP/M-86 operating system diskette, it displays the following message at the top of the screen.

   ```
   TRANSFERRING
   -SUBMIT  .CMD-
   ```

   Then, the following prompt appears.

   ```
   INSERT DESTINATION DISKETTE INTO DRIVE A
   PRESS RETURN TO CONTINUE:
   ```

9. Remove the CP/M-86 operating system diskette from the disk drive and insert the working copy of dBASE II into the disk drive. Then, press **RETURN**.

   PIP1 writes SUBMIT.CMD onto the working copy diskette. When the file is copied, the PIP1 program again displays the following prompt.

   ```
   INSERT SOURCE DISKETTE INTO DRIVE A
   PRESS RETURN TO CONTINUE:
   ```

10. Insert the CP/M-86 operating system diskette into the disk drive again and press **RETURN**. The following message is displayed.

    TRANSFER COMPLETE
    -SUBMIT .CMD-

    The system then displays the following prompt.

    A (AGAIN) OR E (END) ?

11. Press **SHIFT E**. The "A>" prompt reappears.

PUTTING THE LOADER ON THE dBASE II DISKETTE

Your working copy of dBASE II now contains the CPM.SYS and SUBMIT.CMD files as well as the dBASE II system. You must perform one more step before the diskette can be used.

The CP/M-86 operating system diskette includes a program called LDCOPY which moves the system loader routine to other diskettes. If you put the system loader routine on the dBASE II diskette, it will automatically load CP/M-86 into the APC whenever you turn on the power and insert the dBASE II diskette into Drive A.

To use LDCOPY, follow this procedure:

1. Insert the CP/M-86 operating system diskette into Drive A. The "A>" prompt is displayed.
2. Using either uppercase or lowercase characters, type **LDCOPY** and press **RETURN**. When the LDCOPY program is loaded, the following prompt appears.

   LDCOPY VERS 1.0
   Source Drive Name

3. Enter **A**. The program responds with the following prompt.

   Source on A, Then Type Return

4. Press **RETURN**. LDCOPY reads the loader routine from the CP/M-86 operating system diskette, then displays the following prompt.

   Function Complete
   Destination Drive Name (Or Return To Reboot)

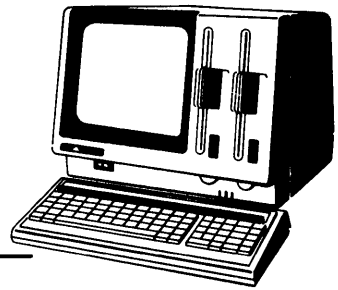
5. The next step depends on the type of APC you have.

   - If you have a dual-drive APC, insert the working copy of dBASE II into Drive B. Enter **B** as the destination drive.
   - If you have a single-drive APC, remove the CP/M-86 operating system diskette and insert the working copy of dBASE II. Enter **A** as the destination drive.

6. The confirmation prompt identifies the destination drive. Make sure the working copy diskette is in the designated disk drive and press **RETURN**. After the system loader routine is on your working copy of dBASE II, the following prompt appears.

   Function Complete
   Destination Drive Name (Or Return To Reboot)

7. Press **RETURN** to end the program. The "A>" prompt reappears.



Chapter 3

A Short Introduction To Using dBase II

Take a few minutes to read through this chapter while you're sitting at the APC, and try the sample commands to see how easy it is to use dBASE II. In this chapter, you'll create a small personnel file and use it to try out some basic commands. Admittedly, this file doesn't contain as many details about each employee as you'd want in a personnel file, and it will consist of just a few employee records. But it will give you a chance to start using dBASE II and, as you'll see later, expanding and changing files are simple processes.

STARTING UP

Each time you use dBASE II, follow these steps to start up the APC.

1. Turn on the APC. If it is already turned on and another program has been in use, turn the APC off and then turn it on again.
2. Insert the working copy of dBASE II into Drive A. Always insert the diskette so that the label faces toward the screen. The label should go in last.

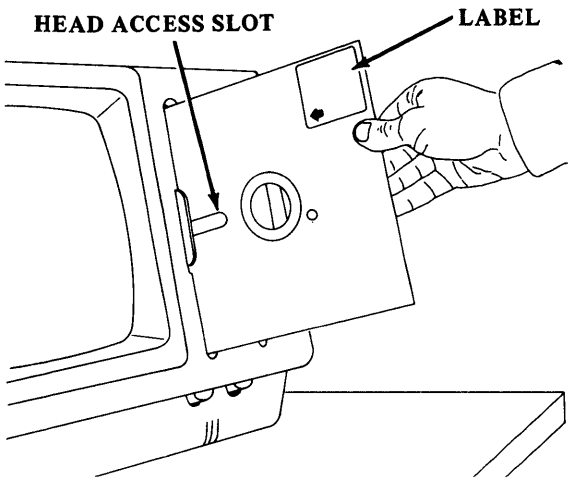
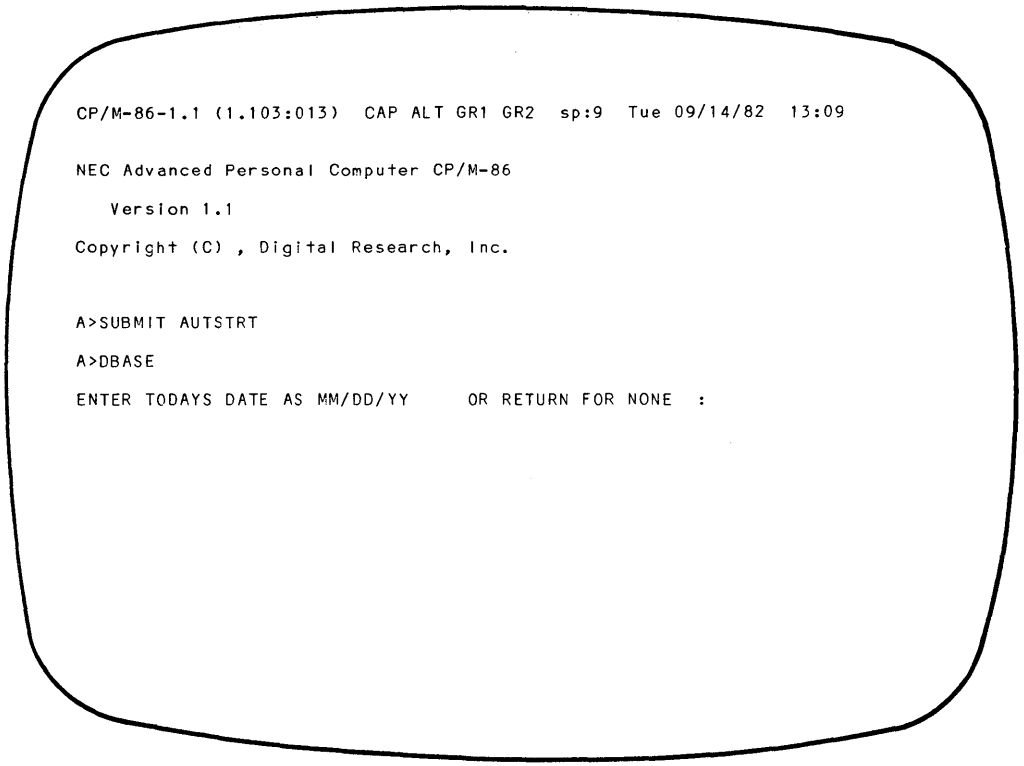


Figure 3-1 Inserting a Diskette

Close the door to the drive. You will hear the diskette spinning as dBASE II loads automatically. Then, you are prompted for the date.



3. For now, simply press **RETURN** to bypass entering the date. The system responds by displaying the message "***dBASE II/86 Ver 2.3B 26 June 82" to identify the version of dBASE II in use. On the next line, the system displays a dot (".") followed by the cursor. dBASE II uses the dot for its prompt. This lets you know that the system is ready to accept commands.

CREATING A DATA BASE DESCRIPTION

Before you can save information in a file, you must name the file and describe the data you want to save in it. It's a good idea to name files and data fields with words that describe their contents. Since you are creating a file of employee records for a personnel system, PEOPLE is a good name for the file. For now, the PEOPLE file should contain the following data fields: name, department number, and salary.

The CREATE command tells the system that you want to describe a new file. At the dot prompt, enter the following command using either uppercase or lowercase

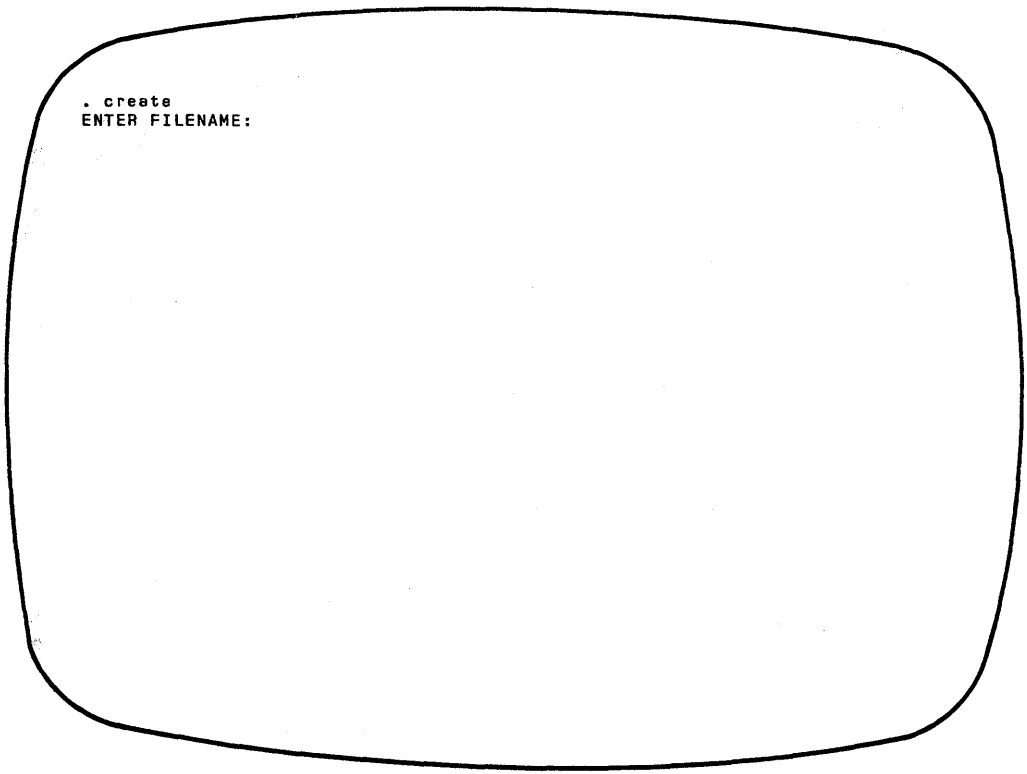
letters, or (if you have installed the function keys) press **PF4**. (Recall that you may use the function keys instead of typing the commands.)

create

NOTE

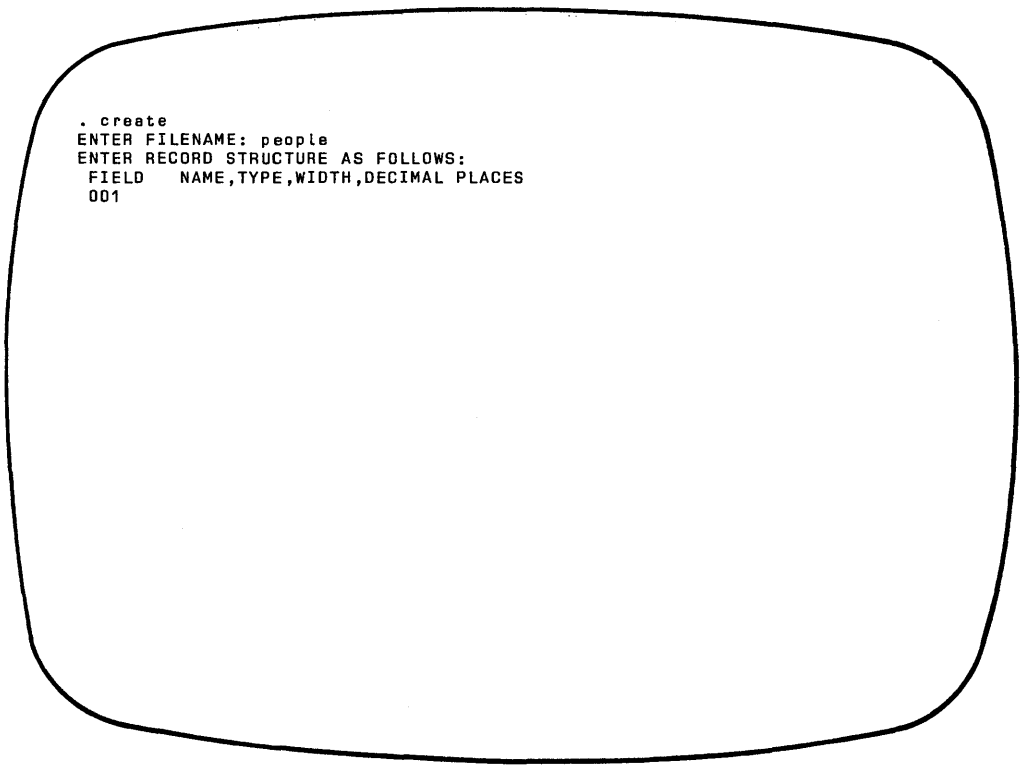
If you discover that you misspelled the command before you press **RETURN**, press the **BACK SPACE** key to position the cursor on the error and retype the word. If you have already pressed **RETURN** when you find the error, press **RETURN** again until the dot prompt reappears. Then, enter the CREATE command again.

The CREATE command initiates a dialog in which the system asks for the information it needs to set up a new file. First, it prompts you for the name of the file.



Enter the name of the file, PEOPLE. Remember, you may use uppercase or lowercase letters when entering all commands and responses to prompts. If you misspell the name, don't worry about it or try to correct it. Just make a note of the name you typed and substitute that name for PEOPLE whenever the file name is needed in the rest of this chapter.

The next prompt requests the characteristics of the data fields, including name, type of data, width (number of characters), and number of decimal places. These fields are explained in Chapter 5.



To complete the PEOPLE file structure, fill in the rest of the screen so that yours looks like the one that follows. In the sample, the words in uppercase type are the prompts that the computer displays. The entries that appear in lowercase and in color are the data you should enter.

CAUTION

Check each line of input carefully against the example before pressing **RETURN**. If you have made an error and already pressed **RETURN**, press **RETURN** again until the dot prompt reappears. Then, reenter the CREATE instruction. This time, when you CREATE the file and name it PEOPLE, the prompt "DESTROY EXISTING FILE? (Y/N)" is displayed. Enter **Y** and continue.

When entering field descriptions, follow these directions.

- Type using either uppercase or lowercase characters.
- Check your entries against the example as each is typed.
- Use the **BACK SPACE** key to position the cursor on errors. Then type over the mistake and reenter the rest of the line again.
- When a line is complete and correct, press **RETURN**. The system prompts you for the next field entry.

Since the data base consists of three fields, press **RETURN** without any data when the prompt for field "004" is displayed. This signals the end of the data base description.

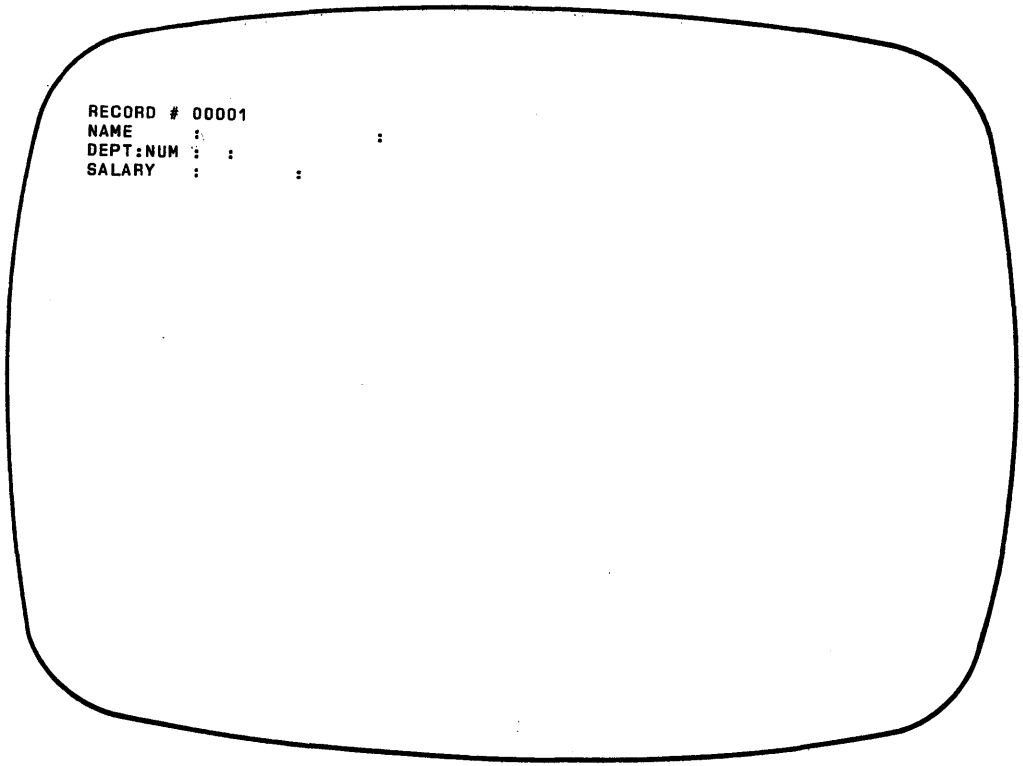
```
. create
ENTER FILENAME: people
ENTER RECORD STRUCTURE AS FOLLOWS:
 FIELD    NAME,TYPE,WIDTH,DECIMAL PLACES
 001      name,c,15
 002      dept:num,c,2
 003      salary,n,8,2
 004
INPUT DATA NOW? Y
```

When description is complete, the system displays the prompt "INPUT DATA NOW?". Press **Y**.

ENTERING DATA RECORDS

The personnel system contains the description of a data base file named PEOPLE.DBF. (*DBF* means data base file. This extension to the file name is explained in Chapter 5.) dBASE II calls the description you entered through the CREATE dialog the *record structure*. Think of it as a framework into which specific values are placed. The system is now in data entry mode, ready to accept name, department number, and salary data for each employee.

In data entry mode, the system displays the structure for one record at a time and waits for input to each field, as in the following screen.



Enter the employee information in Table 3-1 into the PEOPLE data base record structures, one field at a time. Watch the screen as you complete the first few fields. You will notice that sometimes you have to press **RETURN** after entering data and sometimes the cursor jumps to the next field on its own.

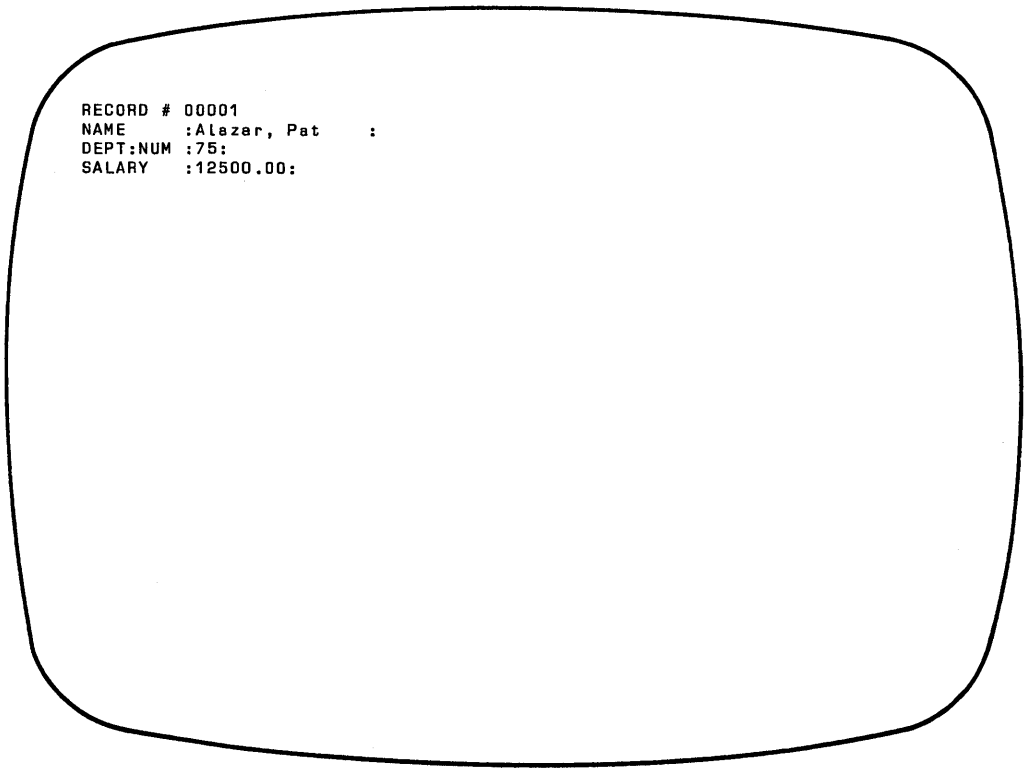
- When the length of a data entry is less than the maximum number of characters allowed for that field, you must press **RETURN** to continue entering data at the next field.
- When the data completely fills a field (like department number) the APC "beeps" to signal that the field is complete and then automatically moves to the next available field.

Table 3-1 Sample Data For PEOPLE Data Base File

NAME	DEPT:NUM	SALARY
Alazar, Pat	75	12500.00
Embry, Albert	89	22200.00
Destry, Ralph	38	15575.00
Howser, Peter	89	9500.00
Clinker, Duane	54	23450.00
Brown, John	54	21000.00

If you misspell a word, don't worry about it now. You can use the **BACK SPACE** key to position the cursor on the error and correct it before going on to the next record, or you can leave the error in the record since this is only a sample data base.

The following display shows the completed data entry screen for the first record.



To stop entering data after the sixth record, press **RETURN** when the cursor is on the first character of the first field of RECORD #00007.dBASE II exits from the data entry mode, returns to command mode, and displays the dot prompt.

NOTE

If you accidentally get back to the dBASE II dot prompt before entering all the data, enter the following two commands to continue entering input with the next record.

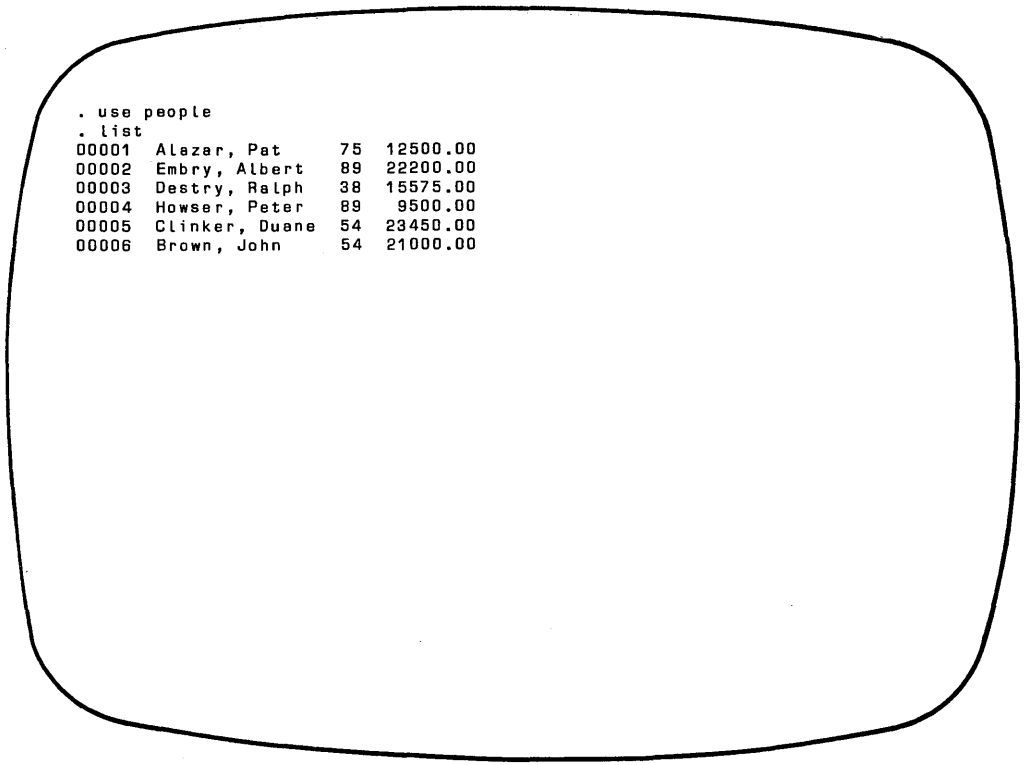
```
use people
append
```

LISTING THE CONTENTS OF A DATA BASE

There are six data records in the PEOPLE file. You can easily display their contents. First, tell the system which file you want to work with by entering the USE command. Then LIST the file. Enter these commands in response to the dot prompt:

```
use people
list
```

The file is displayed, as in the following screen.



Notice that each employee record is preceded by a number. This sequence number was automatically assigned by the system.

SORTING THE DATA BASE FILE

With so few employees, it's easy to find the record of any given person. But suppose the company had 500 or 5000 employees. It would be hard to find a particular employee's record if the records were in random order as they are here. So sort the file into alphabetical order and store the results in a new file named ABCORDER by entering the SORT command as follows:

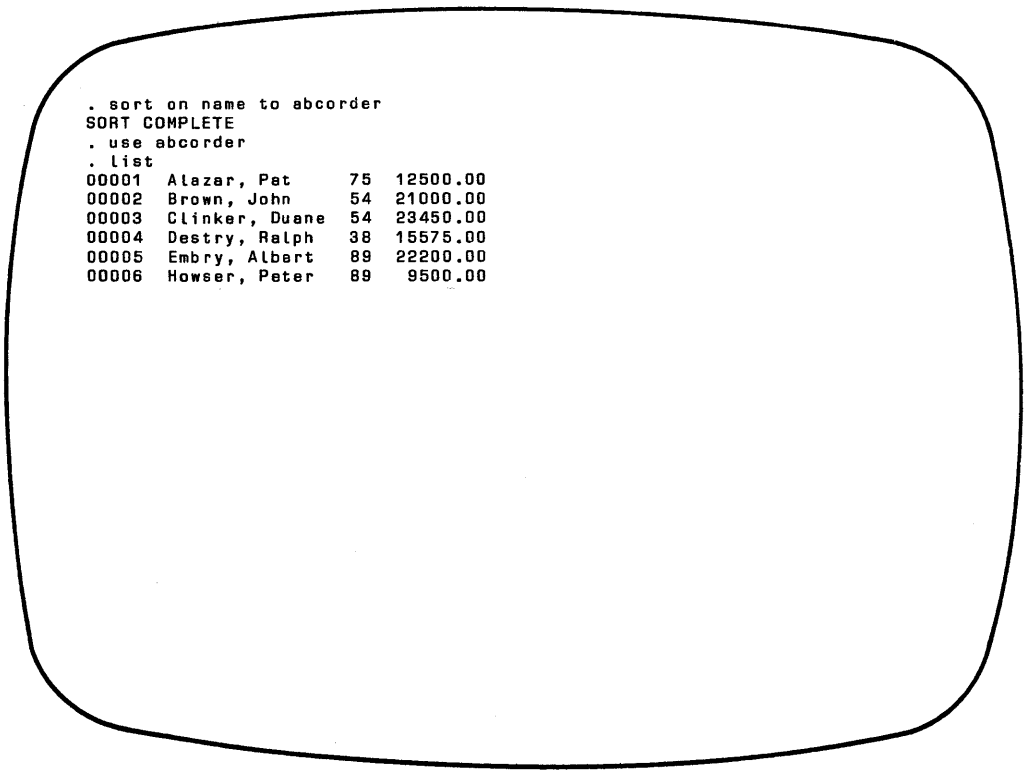
```
sort on name to abcorder
```

You will hear some noises as dBASE II sorts the file. The system notifies you that it has finished by displaying the message "SORT COMPLETE".

Check to see whether the sort worked. Tell the system the name of the file to USE and LIST by entering the following commands:

```
use abcorder
list
```

The file is displayed in alphabetical order on the screen as it appears in the display that follows.



```
. sort on name to abcorder
SORT COMPLETE
. use abcorder
. list
00001   Alazar, Pat      75   12500.00
00002   Brown, John      54   21000.00
00003   Clinker, Duane   54   23450.00
00004   Destry, Ralph    38   15575.00
00005   Embry, Albert    89   22200.00
00006   Howser, Peter    89    9500.00
```

Employee records have been sorted into alphabetical order and saved in a file called ABCORDER.DBF. There are now two copies of the PEOPLE file stored on the diskette. PEOPLE.DBF is the original file. The records are in the order in which they were entered. ABCORDER.DBF consists of exactly the same records but in alphabetical order.

CHANGING THE DATA

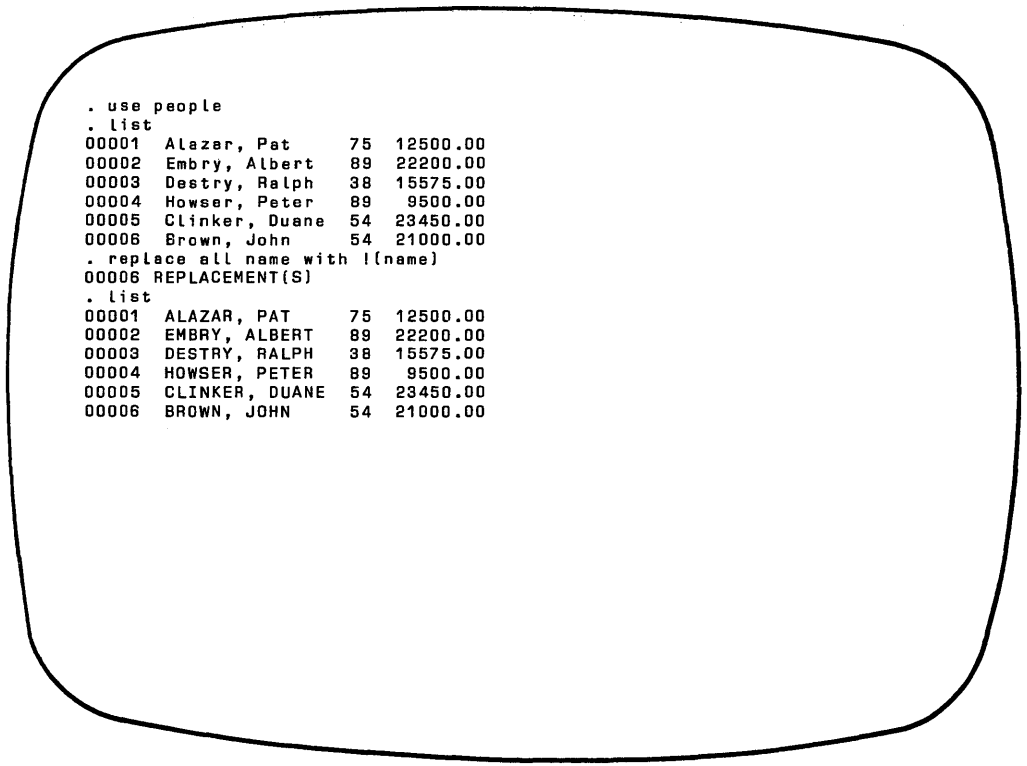
There are many ways to change the information in data base files using dBASE II. One useful command is described in this section. If you typed the sample data exactly as it appeared in the table, names on the PEOPLE file appear with initial capital letters. You can convert the names to all uppercase letters with one command. Enter:

```
use people
list
replace all name with ! (name)
```

The computer performs the replacements to the names right in the PEOPLE file and displays the message "00006 REPLACEMENT(S)". To verify the changes, enter:

```
list
```

The display on your screen looks similar to the one that follows.



```
. use people
. list
00001  Alazar, Pat      75  12500.00
00002  Embry, Albert    89  22200.00
00003  Destry, Ralph    38  15575.00
00004  Howser, Peter    89   9500.00
00005  Clinker, Duane   54  23450.00
00006  Brown, John      54  21000.00
. replace all name with !(name)
00006 REPLACEMENT(S)
. list
00001  ALAZAR, PAT      75  12500.00
00002  EMBRY, ALBERT    89  22200.00
00003  DESTRY, RALPH    38  15575.00
00004  HOWSER, PETER    89   9500.00
00005  CLINKER, DUANE   54  23450.00
00006  BROWN, JOHN      54  21000.00
```

VIEWING THE DATA SELECTIVELY

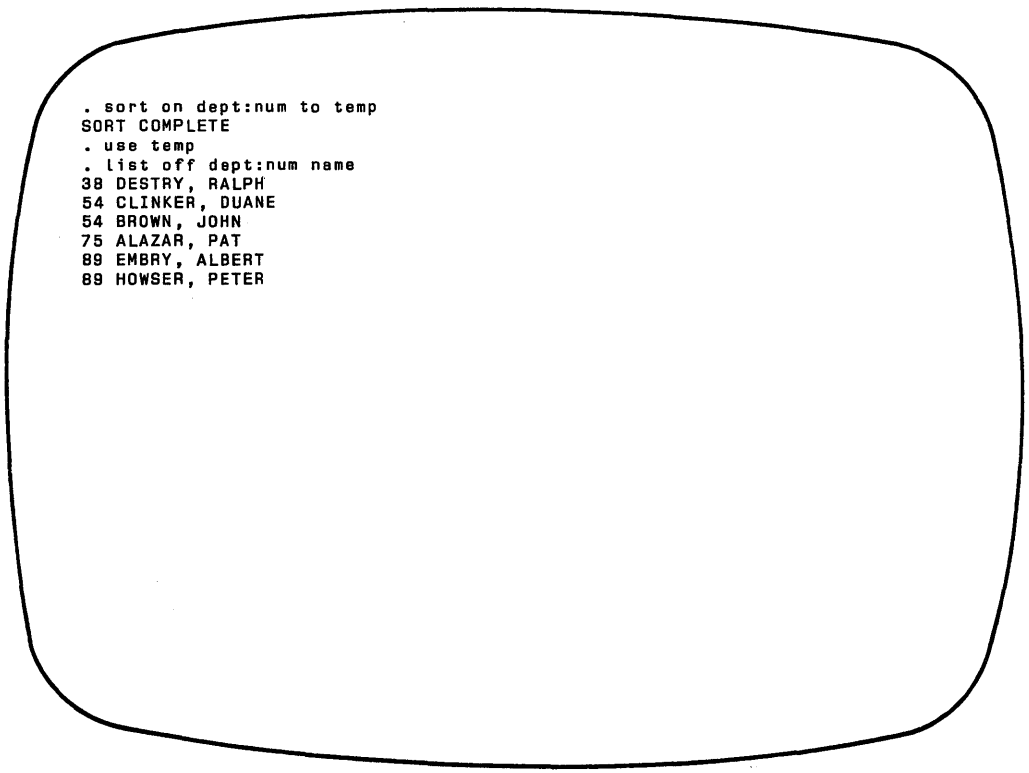
Sometimes you want to see only selected fields from a record and you'd like them displayed in a different order than they appear in the file. To list employees in department number order, first SORT the file on department number by entering the following command.

sort on dept:num to temp

Once the file is sorted, the "SORT COMPLETE" message appears. You can then USE the sorted file and LIST only the fields you want to see. In the following LIST command, the keyword "off" is used to suppress printing the record sequence number. Enter:

use temp
list off dept:num name

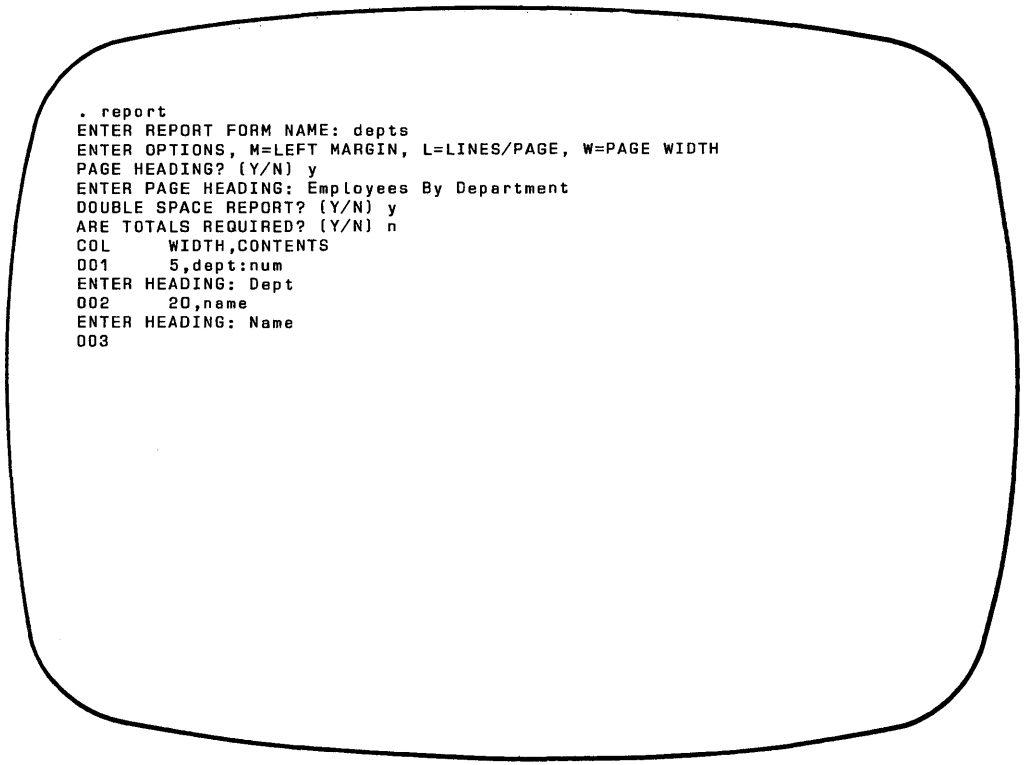
The system lists only the department number and name of each employee.



REPORTING

This list can be made into a report with a title and headings with the REPORT command. Generating a report is similar to creating a file. The system presents a series of questions in order to get the information it needs about the format and content of the report.

The following display screen demonstrates the report formatting dialog. Again, the dBASE II system's prompts appear in uppercase. Your responses are printed in green lowercase type.



```
. report
ENTER REPORT FORM NAME: depts
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH
PAGE HEADING? (Y/N) y
ENTER PAGE HEADING: Employees By Department
DOUBLE SPACE REPORT? (Y/N) y
ARE TOTALS REQUIRED? (Y/N) n
COL     WIDTH,CONTENTS
001     5,dept:num
ENTER HEADING: Dept
002     20,name
ENTER HEADING: Name
003
```

Use the following procedure to produce a sample report.

1. Enter the REPORT command at the dot prompt to start the dialog:

   report

   NOTE

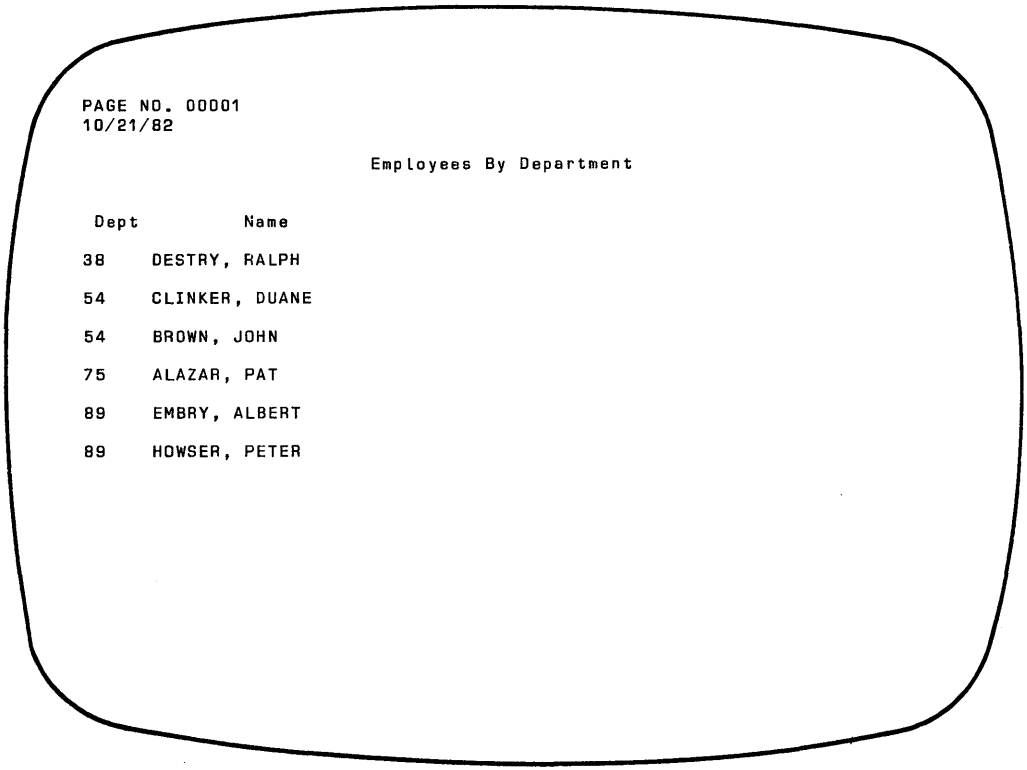
   All dBASE II commands can be shortened to four (or more) letters. Instead of typing the entire word "report", you can enter just the first four letters "repo".

2. Enter the dialog responses as they appear in the previous display. No response is shown for the following two prompts.

   ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH
   003

   Just press **RETURN** for these. In the first case, you've told the system to use the predefined page options. In response to the second prompt, **RETURN** signals the end of the report description.

3. The system displays the report immediately. It looks like the one that follows.



```
PAGE NO. 00001
10/21/82

                        Employees By Department

 Dept          Name

38     DESTRY, RALPH

54     CLINKER, DUANE

54     BROWN, JOHN

75     ALAZAR, PAT

89     EMBRY, ALBERT

89     HOWSER, PETER
```

USING THE PRINTER

The format for the report was saved on diskette in a file named DEPTS.FRM. (The ".FRM" ending is automatically added to the report name. It identifies the file as a form file created by the REPORT command.) This file can be used every time you want the "Employees By Department" report. The report can be displayed on the screen and, if you have a printer, it can also be printed. To see how this works, turn on the printer and line up the paper in it. If you are using single sheets of paper, enter:

set eject off

Enter the report command:

report form depts to print

The printed report should look exactly like the report that was displayed on the screen. In fact, as it is printed it is also displayed on the screen.

ENDING A dBASE II SESSION

To end a dBASE II session, enter the following command:

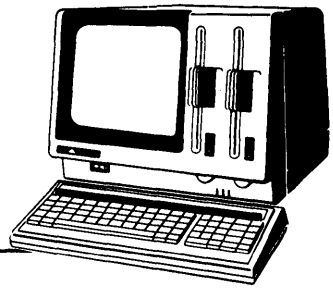
```
quit
```

The message "*** END RUN dBASE II ***" is displayed.

NOTE

QUIT must be entered every time you end a dBASE II session. This automatically closes all files properly.

With the few commands introduced in this chapter you now know enough to CREATE data bases and enter data into them. You can then USE the data base, SORT it, and LIST and REPORT its contents. This chapter has given you an idea of how easy it is to use dBASE II.



Chapter 4

Planning A Data Base

This chapter takes you through the process of determining the contents of the sample data base that is used in this guide. You can skip the chapter and still learn to use dBASE II. However, the material included here defines some basic terms and will answer questions about why the data base files contain the data they do. This should help you in setting up your own data bases.

WHAT IS A DATA BASE?

A data base is an organized collection of data. A data base management system, like dBASE II, is a set of programs that allows you to create, modify, and use data bases to get information. The first step in setting up a data base is to define what information you need to get from it. From that, you can work backwards to specify what the data base should contain and how it should be organized.

A library card catalog is a familiar example of a data base. It contains data about books - titles, authors, publishers, subjects, and so forth. The information is organized. For each book, there is one valid combination of title, author, and publisher. If you want to find a particular book, or information about that book, you can do so because the card catalog is indexed by author, title, and subject. It's convenient to picture the card catalog as a table.

Table 4-1 Card Catalog

BOOKS				
TITLE	AUTHOR	SUBJECT	PUBLISHER	REF #

SOME DEFINITIONS

dBASE II looks at data in much the same way as the card catalog in Table 4-1. Data base files are organized as two-dimensional tables with rows and columns of data. The column headings are *fields* and they name the data items. Each row describes one particular entry in the table. In the card catalog, each row describes a book. A row is a *record* in the data base file. The complete collection of records is called a *file.*

A card catalog *data base* actually consists of three files. The data in the files are identical, but the order of the fields (and therefore the organization of the files) differs. One file is sequenced by author. Another is arranged by title. The third is organized by subject.

Like the card catalog, the records in a dBASE II "BOOKS" data base can be accessed by title, author, or subject. Unlike a manual system, however, the dBASE II computer programs do not need three different physical files to provide three different accesses to data. dBASE II *indexes* the file on three *keys.* Each key is a field that can be used to identify a particular record or records. dBASE II files can be indexed on a field that has unique values, such as title, or on a field such as author that can have the same *value*, the author's name, in more than one record.

Because the records in a data base file can be indexed on more than one key, the data base can consist of only one file. There is no redundant data. If the data base needs to be updated when a book is added or deleted, only one file has to change. Since there can be more than one key, there is no way to arrange the file in any key sequence order. Therefore, the order of the columns and rows is not important to dBASE II. It is good practice, however, to be sure that no two rows contain exactly the same data.

Often, more than one data base file is needed in a data base. For instance, if the "BOOKS" file were used in a mail-order business, the user would probably also need access to data base files for inventory, customers, sales tax, and so on. All of these separate files, taken together, would be considered the *data base.*

A SAMPLE DATA BASE

The demonstration data base in this guide stores the information for a personnel system. The files are set up to provide the following information.

- Lists of employees sorted by department or job title
- Average salary for all employees in a department
- Salary range for any job title

- Salary for each employee
- Home address for each employee
- Status (active or inactive) for each employee
- Number of employees with salaries above and below the midpoint of the range for each job
- Total of annual salaries by department and for the entire company.

To respond to these information needs, the personnel data base should have the following information.

- Employee's name
- Employee's address
- Employee's job title
- Employee's department
- Employee's status
- Employee's salary
- Salary range for each job title

The data base could be set up as one file with a record for each employee. Each record would consist of all the fields listed above, and could be represented in the format of Table 4-2.

Table 4-2 Employees Data Base File

EMPLOYEES							
NAME	ADDRESS	TITLE	DEPARTMENT	SALARY	STATUS	LOW SALARY	HIGH SALARY

The TITLE field would contain the same value for all employees with the same job title. The LOW and HIGH SALARY fields would also be the same for all employees with the same title. This data base structure creates a lot of redundant data, which takes up space on a diskette and may be unnecessarily difficult to maintain.

Splitting Into Files

To avoid these complications, it is often easier to divide the data into more than one file. Notice that the personnel data falls naturally into two categories: information about employees and information about jobs. It can be represented in two tables.

Table 4-3 Employees Data Base File

EMPLOYEES					
NAME	ADDRESS	TITLE	DEPARTMENT	SALARY	STATUS

Table 4-4 Job Details Data Base File

JOB DETAILS		
TITLE	LOW SALARY	HIGH SALARY

The two files have one field, TITLE, in common. This allows you to combine the information in the JOB DETAILS file with the information in the EMPLOYEE file based on an employee's title.

Notice how much easier it is to make changes affecting many employees when there are two files. You can change the low and high salary figures for any job title just once, in the JOB DETAILS file, and then use the new figures for all people to whom they apply in the EMPLOYEE file.

These two tables now represent the data for employees and job details as requested and with a minimum of redundancy. However, they can be refined even more.

Refining the Fields

For instance, exactly what does ADDRESS mean? If it will only be printed on reports, it may be all right to think of an address as a single field, like "44 Braddock

St., Hackensack, NJ 07601". But since it really consists of four separate fields, you'd be better off defining it that way. Then, if you need to break the address into three or four lines (to print mailing labels, for example) the data will already be accessible to you in that form. So replace ADDRESS with STREET, CITY, STATE, and ZIP CODE in the file.

The same logic, setting up the data for a variety of accesses, can be applied to the NAME field. You may need to sort the file alphabetically by last name for reports and also print name tags with the employees' first names first. It's a good idea to think of NAME as FIRST NAME and LAST NAME.

NOTE

Dividing one field into several smaller fields does not change the amount of space the system uses to store data on the diskette. It does increase the ease of accessing and manipulating part of a conceptually larger field.

Saving Space

TITLE appears in both the EMPLOYEE and JOB DETAILS tables. That means the field is included in every employee's record. Most likely, many employees in a company have the same job title. The TITLE field must be large enough to hold a meaningful job title, and that results in making the EMPLOYEE records unnecessarily long.

One alternative is to assign each TITLE a code. The JOB CODE would be used in both the EMPLOYEE and JOB DETAILS files, but the corresponding description would be used only in the JOB DETAILS records. This arrangement saves space in the files, and the space savings become more significant as the number of employees with the same title increases. Since dBASE II commands can be used to combine data from two or more data base files, replacing TITLE with JOB CODE in the EMPLOYEE file and adding JOB CODE to the JOB DETAILS file actually makes the data base smaller without losing any information.

Tables 4-5 and 4-6 incorporate these changes to the personnel data base. In this format, the data base is complete, efficient, and low in redundancy.

Table 4-5 Employee Data Base File

EMPLOYEES									
FIRST NAME	LAST NAME	STREET	CITY	STATE	ZIP CODE	JOB CODE	DEPT NUM	SALARY	STATUS

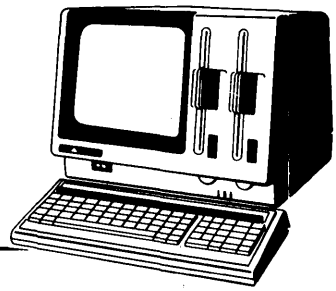
Table 4-6 Job Details Data Base File

JOB DETAILS			
JOB CODE	TITLE	LOW SALARY	HIGH SALARY

Changing The File Contents

There is another advantage to setting up the personnel data base as two files. When you discover the need for a new data field, you can add it to the appropriate file easily. For instance, suppose you decide to include prerequisite skill data as part of JOB DETAILS. You can add the field to the end of the table (file) and not disrupt either the existing JOB DETAILS records or the EMPLOYEE file. Similarly, an addition to the EMPLOYEE record structure (such as month and year hired) does not affect the JOB DETAILS file.

The goal in setting up a data base is to include all the data needed with a minimum of duplication. The best data bases are organized as one or more files, each of which consists of fields describing the properties of only one entity - employees, salary history, payroll figures, job descriptions, and so on. The files should be related to each other as necessary by a limited number of common fields, the way job code links each employee record to a record in the job details file.



Chapter 5

Creating A Data Base

This chapter will get you started creating dBASE II files, entering data, and editing your input to correct errors. In Chapter 3, you performed some of the processes described here. This chapter describes the commands in detail.

GETTING STARTED

Each time you use dBASE II with the APC, follow these steps to start up the system.

1. Turn on the APC. If it is already turned on and another program has been in use, turn the APC off and then back on.
2. Insert the working copy of dBASE II in Drive A with the label side of the diskette facing the display screen.

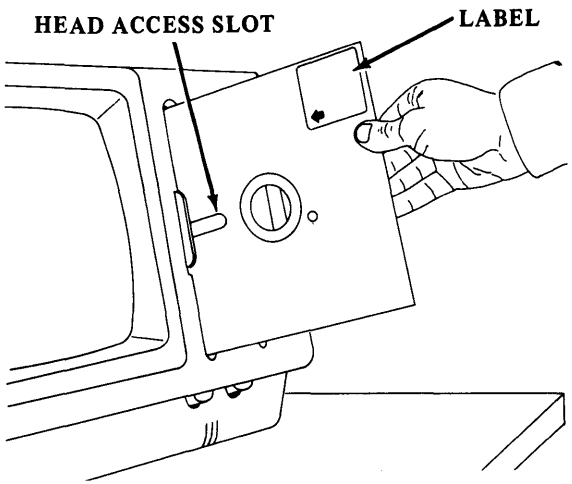


Figure 5-1 Inserting a Diskette

3. Squeeze the diskette drive door shut.

Setting the Date

The dBASE II program loads automatically, and starts execution with a date request.

To bypass the date routine, press **RETURN**.

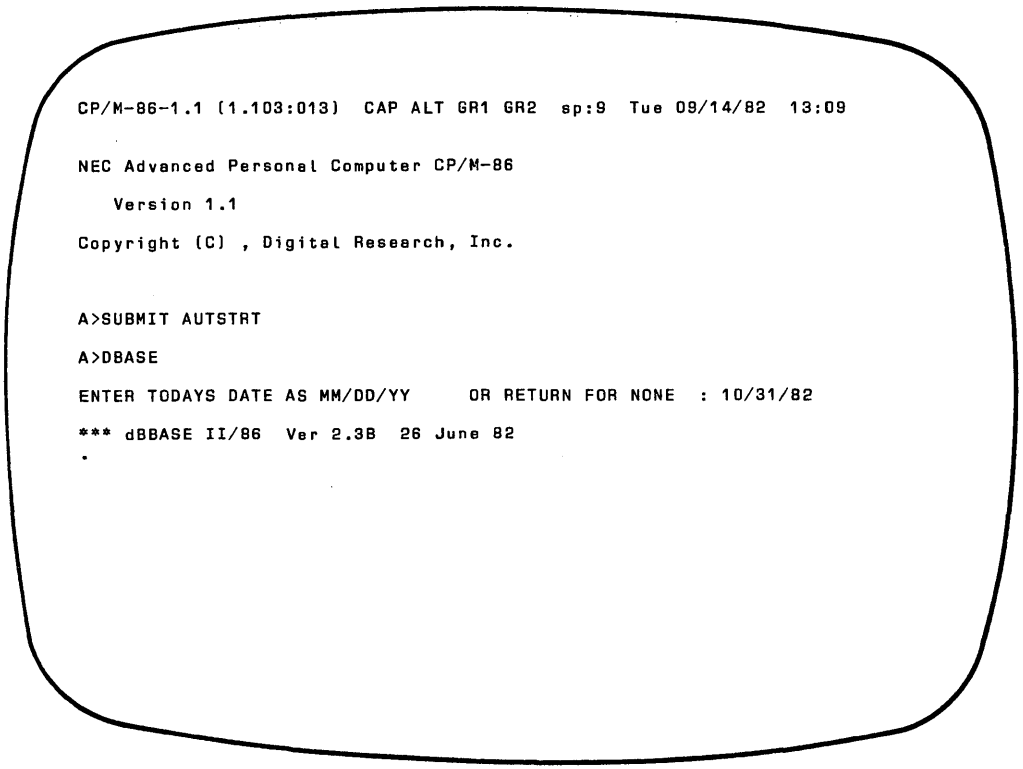
To enter the date, type a valid date (using two digits each for the month, the day of the month, and the last two digits of the year) and press **RETURN**. Use a slash (/) or any other special character (that is, any character that is not numeric or alphabetic) except a period to delimit the date fields. For example, all of the following are valid dates.

09 23 48
10,15,68
07/07/40

The date is checked for calendar accuracy. Invalid dates are rejected (except February 29, 1900 and February 29, 2100). The date is used by the system in two ways.

- When records are added to or deleted from any file, the date you entered is recorded in the file. This is useful for keeping track of when changes were made. If you bypass date entry during sign-on, the file's date of last update is set to zeroes when a new file is created and is left unchanged for updates to existing files.
- The date is printed in report headings for any report generated during the session.

When the date has been accepted or bypassed, the dBASE II sign-on message is displayed, naming the version of the system that is in use. Then, the dBASE II command mode prompt (".") appears on the next line of the screen, indicating that the system is ready to accept commands, as in the following screen display.



Using dBase II Data Diskettes

If you have a single-drive system, the dBASE II programs and files are always on the same diskette in Drive A.

If you have a dual-drive system, you have the option of using one diskette drive, Drive A, for the dBASE II program diskette and the second drive, Drive B, for a data diskette. This setup is useful when you anticipate having very large data bases.

Data diskettes must first be initialized using the FORMAT program (see Chapter 2). Be sure to label the diskettes accurately. No other diskette preparation is necessary.

To use a data diskette in Drive B, insert the diskette in the drive and close the drive door. Since the system assumes that data files are on Drive A, called the *default drive*, you must tell the system that it is to read data from and write data to the diskette in Drive B. There are two ways to do this.

- The easiest way is to change the default drive from Drive A to Drive B with the following command:

  set default to b

  The system will perform all read and write functions on the files in Drive B until the default is reset with the command "set default to a" or you end the dBASE II session.

- If some data files are on the diskette in Drive A and others are on the diskette in Drive B, set the default to the drive having most of the files. Then, during the dBASE II session, refer to the files on the default drive by name only (for example, PEOPLE or DEPTS.FRM). Refer to files on the other drive by prefixing the drive specifier and a colon to the file name. For example, if the default drive is Drive A and you want to use a file called NAMES on the diskette in Drive B, you would refer to the file as B:NAMES.

NOTE

All references to files on the alternate drive must be prefixed by the drive specifier and a colon.

CREATING A DATA BASE

Data base files consist of two parts. The *record structure* describes the data. The *data records* contain the data values. The first step in setting up a data base file, determining the structure, was described in Chapter 4. Once you have decided what information to include in a file and what format that data will have, you can create the file.

Naming the File

Every dBASE II file must be named. Select *file names* that are as meaningful as possible within the following limitations.

- Use from one to eight characters.

- All letters, digits, and special characters may be used except those having special meaning to the CP/M-86 command processor. DO NOT USE A SPACE OR ANY OF THE FOLLOWING SPECIAL CHARACTERS IN FILE NAMES.

  < > . , ; = ? * [ ]

- Although dBASE II accepts embedded colons (":") in a file name, it is recommended that this symbol be used only to specify an alternate drive (for example B:PEOPLE).
- Use a letter as the first character of a file name.

NOTE

For a complete discussion of file naming conventions, see the *CP/M-86 System User's Guide*.

JOBDET FILE

In this chapter, you will create the JOB DETAILS file described in Chapter 4. To do so, enter the following command at the dot prompt:

create jobdet

You can include the name of the file in the command or just enter the word CREATE. Both forms are acceptable. When the file name is omitted, the system prompts you for it. In either case, dBASE II creates a file called JOBDET.DBF on the diskette in the default drive.

NOTE

If you try to CREATE a file that already exists, dBASE II asks "DESTROY EXISTING FILE?". You must press either **Y** or **N** to continue.

If you press **N**, the named file is not created. Select another name for the file and execute the CREATE command again.

If you press **Y**, the existing file is destroyed and replaced by the new file.

DBF EXTENSION

The ending ".DBF", short for data base file, is automatically added to the file name. It tells both the dBASE II programs and CP/M-86 what the file *type* is. When you ask for a list of files on a diskette, this ending identifies dBASE II *data* files.

Describing the Record Structure

Once dBASE II has accepted the file name, it prompts you for the names and characteristics of the data items that make up the records, as in the following display.

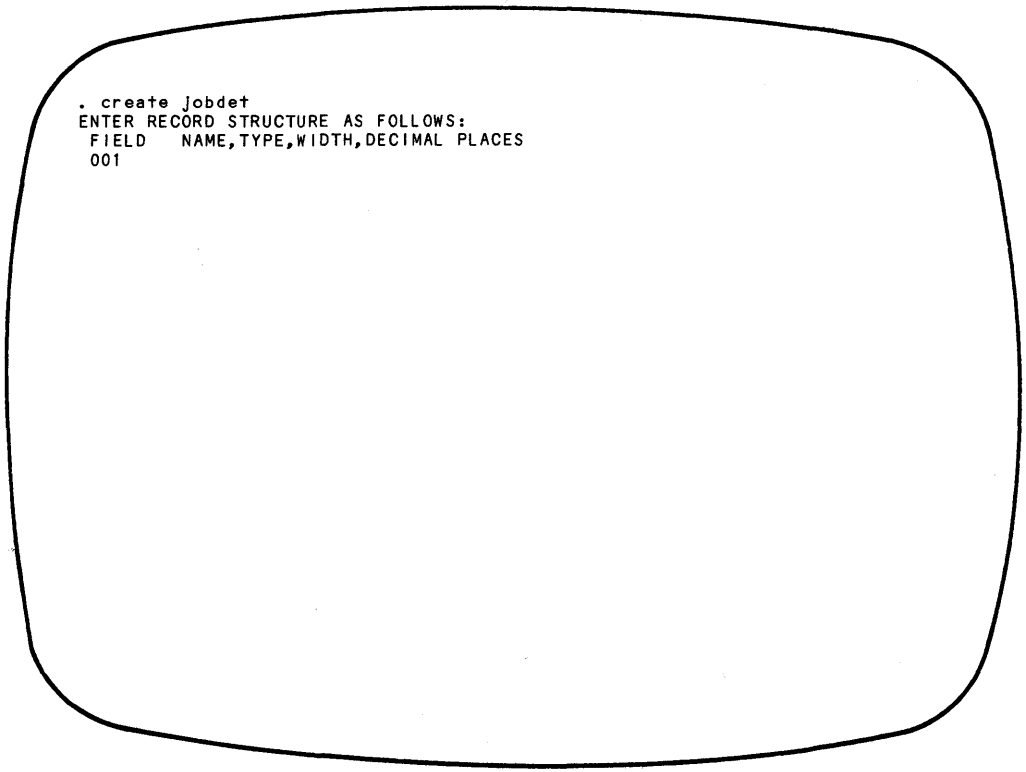


Table 5-1, displays the fields for the JOB DETAILS file developed in Chapter 4.

Table 5-1 Job Details Data Base File

JOB DETAILS			
JOB CODE	TITLE	LOW SALARY	HIGH SALARY

Data base files are best pictured as two-dimensional tables. The column headings and some additional information to describe the field characteristics are the *record structure*. This is essentially a map of the data record format. Using dBASE II, each data base structure can contain descriptions for up to 32 fields.

DATA FIELDS

Each entry in the structure refers to a field in the data records. A field is defined by four characteristics: name, type, width, and decimal places.

Field Name is the name by which each data field is referenced in all operations. It must conform to the following rules.

- Up to ten characters
- Letters, digits, and colon (:) only
- Uppercase and/or lowercase letters acceptable
- Must begin with a letter

Type is the kind of data within each data field. dBASE II uses three symbols to specify the type of data within a field.

- C = Character

  The field can contain all printable ASCII characters (letters, integers, and space).
- N = Numeric

  Only digits, the plus and minus signs, and the decimal point are valid field entries. Numeric fields are accurate to ten places, including the sign and the decimal point.

- L = Logical

  Logical fields hold True or False values. Acceptable values are Y, y, T, and t True and F, f, N, and n for False.

Width is the number of character positions needed to contain the data that will be placed into the field.

- Character fields can be from one to 254 characters.
- Numeric fields can be a maximum of ten positions, including the sign and point.
- Logical fields always have a width of one character position.

Decimal Places tells the number of positions to the right of the decimal point. This characteristic only applies to numeric fields and should be bypassed for character and logical data fields. It can also be omitted for whole number fields, in which case the number of decimal places is assumed to be zero.

You do not have to supply values for all fields in a data record. dBASE II automatically fills in a *default value*, based on the field type, when the value is omitted.

Table 5-2 summarizes the record structure characteristics for each type of data.

Table 5-2 Record Structure Values By Data Type

DATA TYPE	MAXIMUM FIELD WIDTH	DECIMAL PLACES	DEFAULT VALUE
C	254	N/A	space
N	10	0-9	0
L	1	N/A	F

COMPLETING THE SAMPLE STRUCTURE

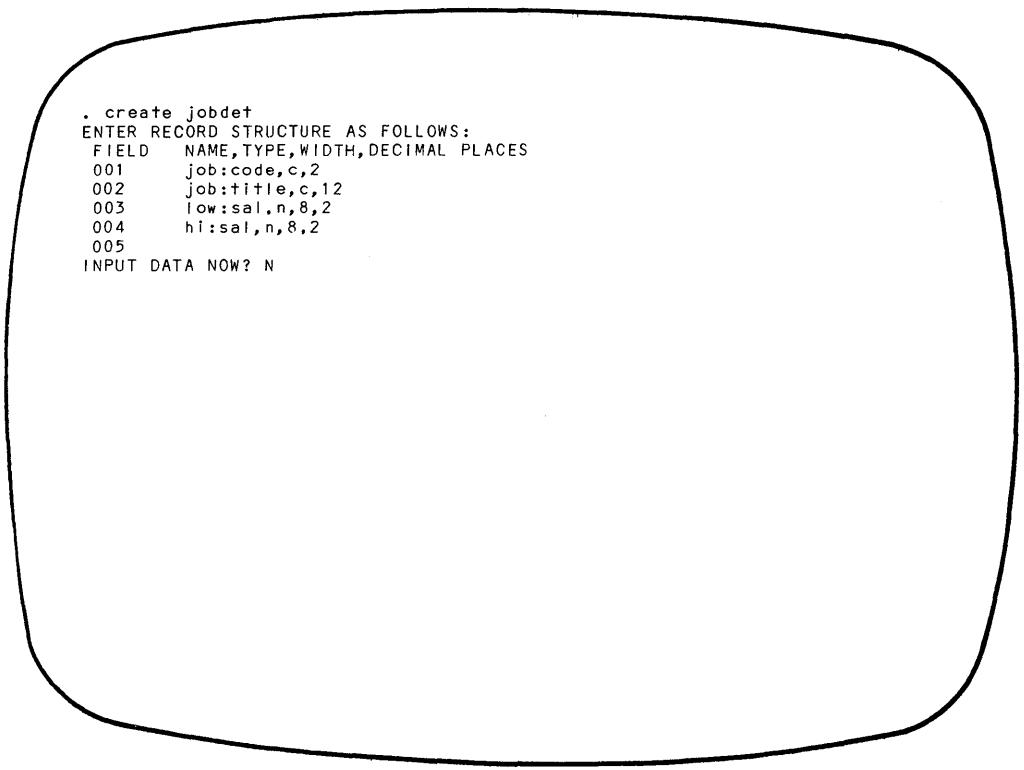
JOBDET consists of four fields. Code is a two-character field consisting of one letter and one digit. Title will contain the name of each job in the company. A title can consist of both letters and numbers. Fifteen spaces should be enough for each title. The low and high salary fields are numeric and will be dollar and cents values. The maximum salary allowed will be 99999.99. Table 5-3 summarizes the data field characteristics to enter in the JOBDET record structure.

Table 5-3 JOBDET Record Structure

FIELD NAME	TYPE	WIDTH	DECIMAL PLACES
JOB:CODE	C	2	
JOB:TITLE	C	15	
LOW:SAL	N	8	2
HI:SAL	N	8	2

Enter the structure for the JOB DETAILS file so that your screen looks like the completed example.

Use a comma to separate the characteristics of each field. Check your input for accuracy before pressing **RETURN** at the end of each line. If you find an error, correct it by using the **BACK SPACE** and cursor movements keys to position the cursor on the error and reenter the correct values. Do not enter any data for field 005. Instead, press **RETURN** to signal the end of data.



In response to the question "INPUT DATA NOW?", press **N** (for "NO").

NOTE

If you press **Y**, the system automatically puts you in data entry mode. To return to command mode, press **RETURN** again.

Saving the Description

The JOBDET.DBF file is set up and can accept data now or at any time in the future. In Chapter 3, you entered data immediately after creating the structure. This time, since you responded "N" to the question "INPUT DATA NOW?", the system did not go directly into data entry mode. Instead, dBASE II saved the file structure and remained in command mode. You can continue with any dBASE II command or even QUIT the system and turn off the computer. When you want to add data to the file, just tell the system the name of the file to USE and the function to perform on that file.

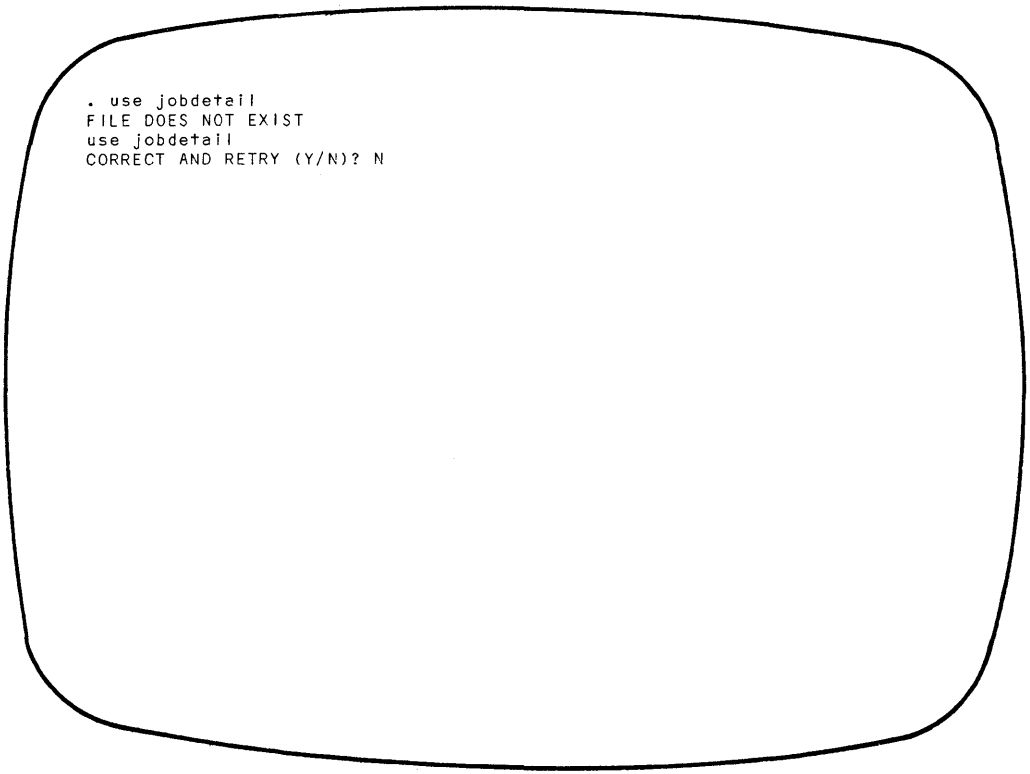
Using Files

The USE command specifies which preexisting data base file is to be the file in USE. You must USE a file to access and manipulate the data it contains. Usually, dBASE II allows only one file to be open and available to you at a time. Therefore, when you issue a USE command, the first thing the system does is close any file that is already open.

Then, if you include a file name in the command (for example "USE PEOPLE" or "USE JOBDET"), the named file is opened. From then on, until another USE command is entered, dBASE II executes all commands on the file currently in USE.

If you enter the USE command without naming a file (that is, you enter only the one word "USE") the system closes any open file but does not try to open a new file.

If you enter the USE command with the name of a file that the system doesn't recognize (probably due to a misspelling), dBASE II responds with the prompt shown on the next screen.



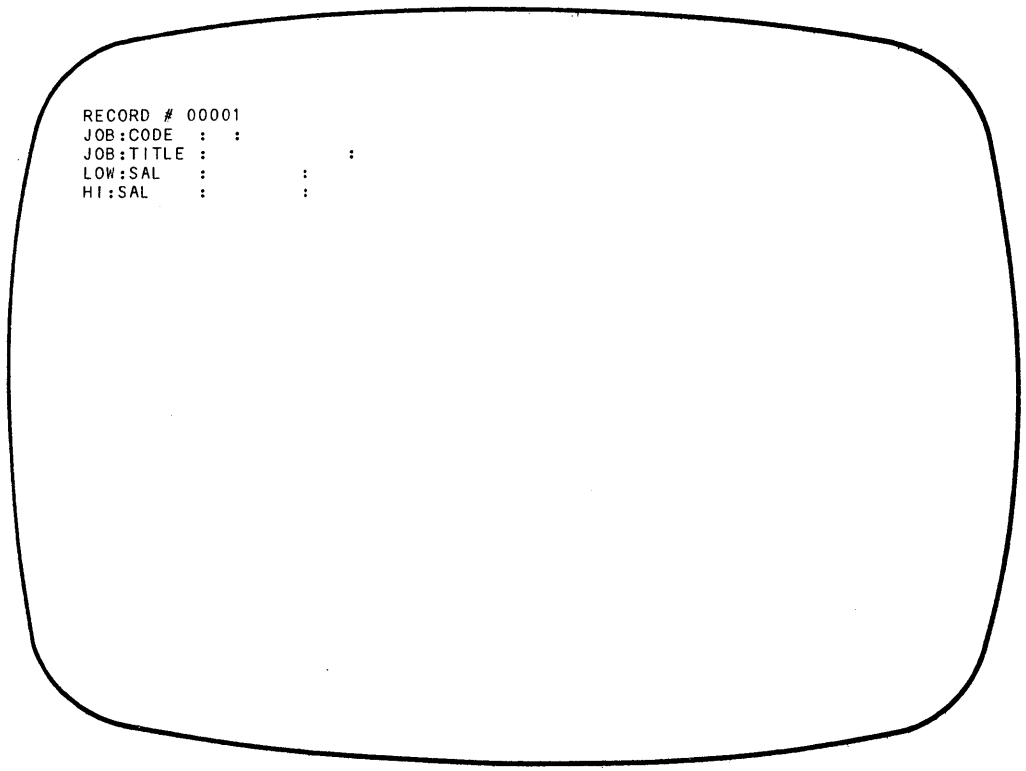
For now, press **N**. Check your spelling and then reenter the USE command.

Entering Data Records

Each data base file may consist of from 0 to 65,535 data records. The records contain the values for the fields described in the structure. APPEND adds records to the end of the data base in USE. To create JOBDET data records, enter:

```
use jobdet
append
```

The screen is erased. Then, the system displays the first available record number and the record structure. All data field names are displayed starting in the upper left corner of the screen. The cursor is positioned at the first character of the first field. Field lengths are indicated by the number of spaces between the colons.

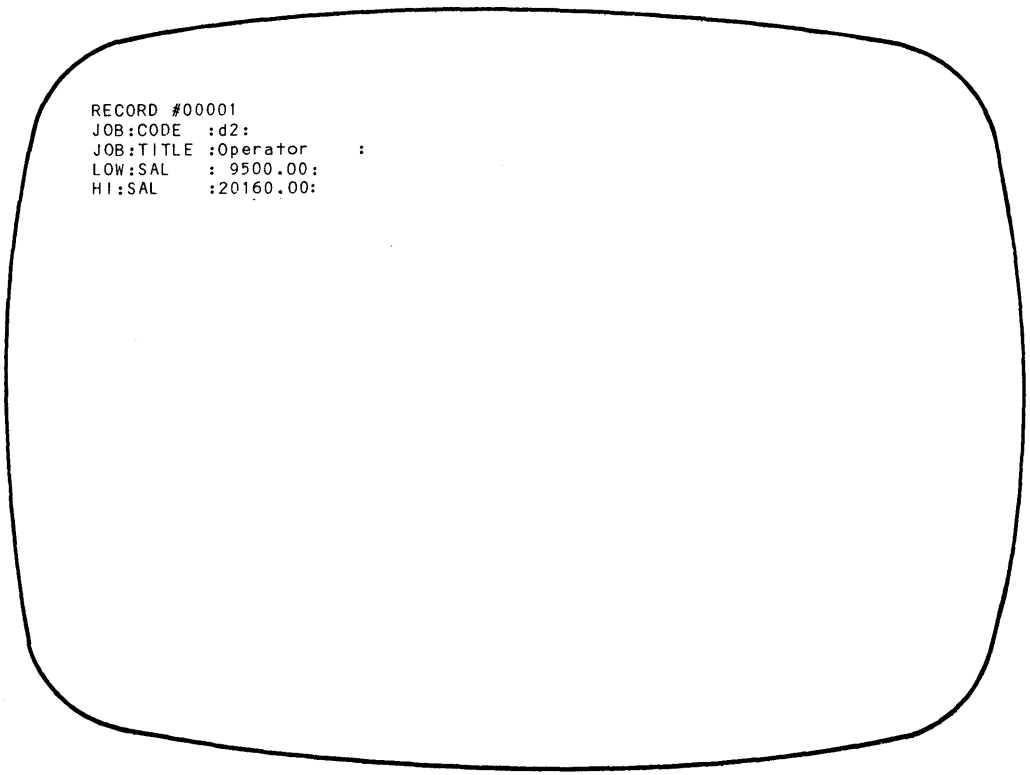


Enter the records defined in Table 5-4. Type carefully, but don't worry about misspellings. If you find an error in the record you are working on, you can correct it by using the **BACK SPACE** and cursor movement keys to position the cursor on the error and retyping the entry.

Table 5-4 JOBDET Record Values

RECORD #	JOB:CODE	JOB:TITLE	LOW:SAL	HI:SAL
00001	d2	Operator	9500.00	20160.00
00002	d8	Shift Leader	15000.00	26750.00
00003	ml	Team Leader	12000.00	15500.00
00004	m9	Dept Manager	20000.00	35000.00
00005	p3	Programmer	15750.00	25000.00
00006	p8	Sr Programmr	20000.00	28000.00
00007	p9	Sr PrgAnlyst	27500.00	36000.00
00008	s8	Typist	9500.00	15750.00

The following screen shows how your display should look when RECORD #00001 is complete.



In data entry mode, when the last field of the current record is filled, dBASE II automatically saves the record and sets up the screen display to accept data for the next record. To terminate data entry after RECORD #00008, press **RETURN** when the cursor is on the first position of the first field in RECORD #00009.

DEFAULTS

If there is no data for a field (except the first field in a record), press **RETURN** to move to the next field. dBASE II automatically supplies values, called defaults, as follows:

- Character fields are filled with blanks;
- Numeric fields are filled with zeroes;
- Logical fields are assigned the value ".F.".

For a numeric field, if no digits follow the decimal point there is no need to type it. dBASE II automatically supplies the decimal point and the correct number of following zeroes, as specified in the "DECIMAL PLACES" field in the record structure.

NOTE

When the cursor is positioned on the first field of a record, pressing **RETURN** without any data has a special meaning to the system. It signals the end of data entry, and the system returns to control mode.

VIEWING THE RESULTS

The LIST and DISPLAY commands can be used to see all or part of the data. In the examples that follow, LIST is used. DISPLAY can be substituted for LIST in all commands, however the number of records that appear on the screen may differ depending on the command used. (The differences between LIST and DISPLAY are explained in Chapter 6.)

Listing Records

It is a good idea to look over your data once you type it in to be sure that it is correct. Enter:

list

The system LISTs the contents of the file that is in USE.

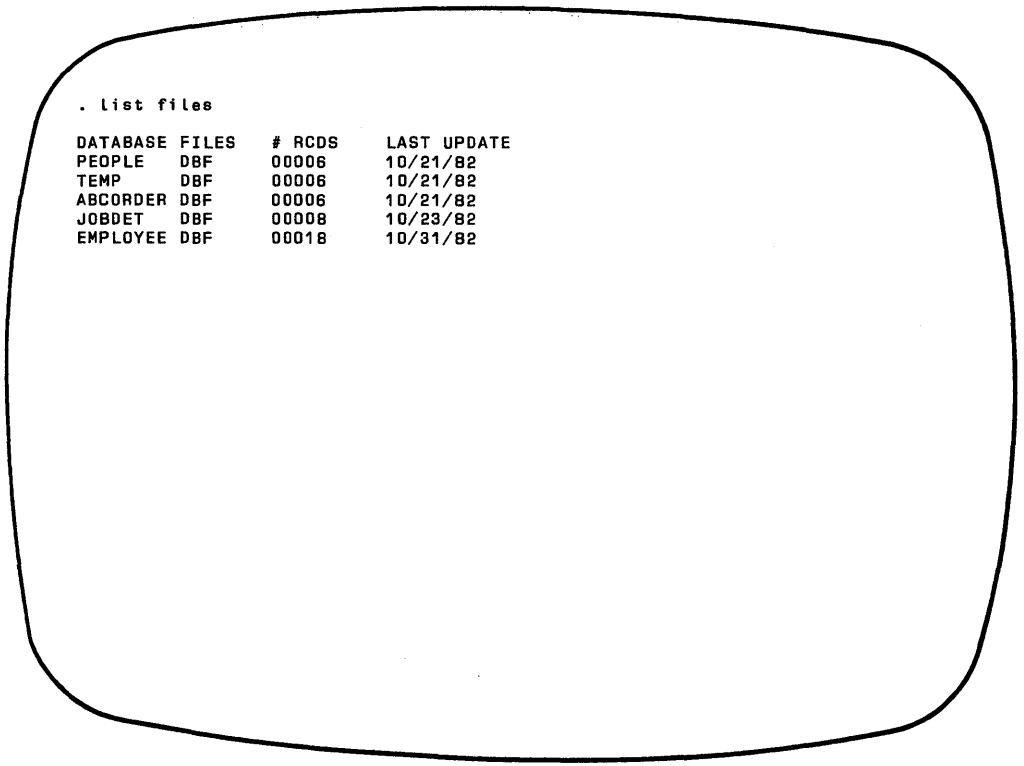
```
. list
00001   d2 Operator        9500.00   20160.00
00002   d8 Shift Leader   15000.00   26750.00
00003   m1 Team Leader    12000.00   15500.00
00004   m9 Dept Manager   20000.00   35000.00
00005   p3 Programmer     15750.00   25000.00
00006   p8 Sr Programmr   20000.00   28000.00
00007   p9 Sr PrgAnlyst   27500.00   36000.00
00008   s8 Typist          9500.00   15750.00
```

Listing DBF File Names

LIST and DISPLAY can be used to find the names of *files* on a diskette. Some characteristics of the files also appear with the file names. This is useful when you have many dBASE II diskettes, and when you forget the names and spellings of file names. Enter:

```
list files
```

The display looks similar to the following screen.



The full names of all files with the "DBF" extension are listed under the heading "DATABASE FILES". When you USE a data base file or otherwise refer to it in commands, omit the extension.

The count of data records in each file is given in the column titled "# RCDS".

The date each file was last updated (if a date was entered at sign-on) appears under the heading "LAST UPDATE". This field is zeroes if no date was entered when the file was created and updated.

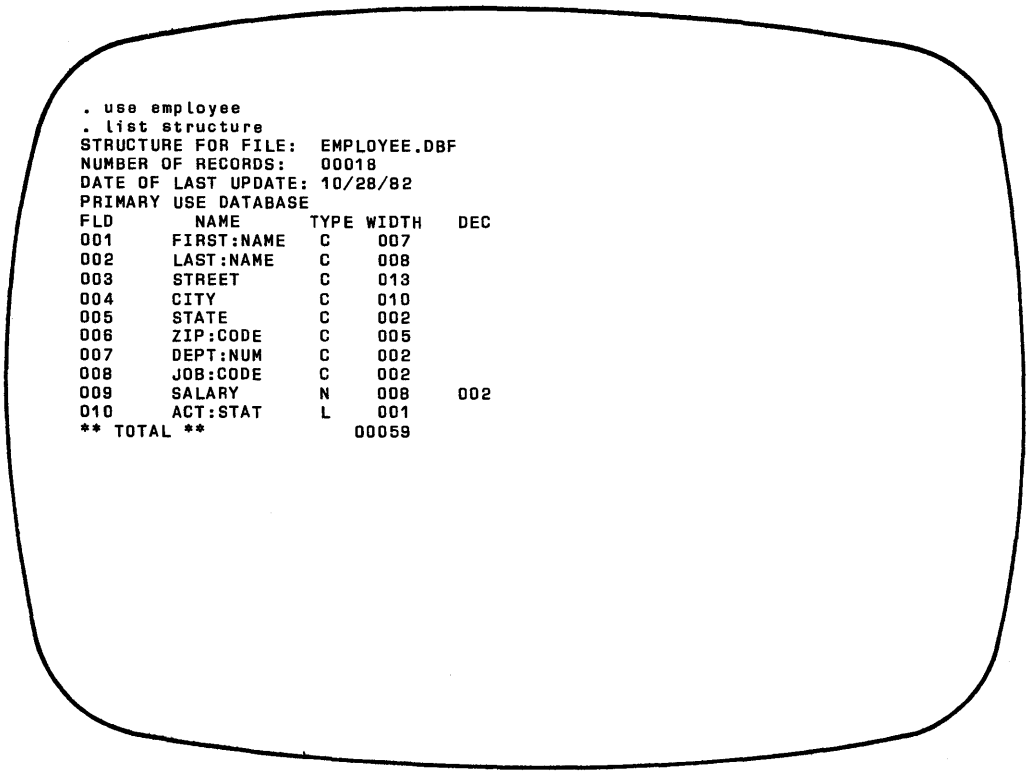
Notice that the display includes the files you created (PEOPLE.DBF, ABCORDER.DBF, TEMP.DBF, and JOBDET.DBF) . At least one other file is also listed. EMPLOYEE.DBF is a sample file that is part of the personnel system used in this guide. This file was included on the distribution diskette so that you would not have to type in all the data. It is an expanded version of the PEOPLE.DBF data base you created in Chapter 3. It consists of the data fields described in Chapter 4 and 18 sample data records.

You can USE EMPLOYEE and LIST the file to see its contents.

Listing the Structure

LIST can also be used to view the record structure. If you do not have a file in USE, select one and issue the USE command for that file. Then, enter:

```
list structure
```



```
. use employee
. list structure
STRUCTURE FOR FILE:  EMPLOYEE.DBF
NUMBER OF RECORDS:   00018
DATE OF LAST UPDATE: 10/28/82
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH    DEC
001      FIRST:NAME   C     007
002      LAST:NAME    C     008
003      STREET       C     013
004      CITY         C     010
005      STATE        C     002
006      ZIP:CODE     C     005
007      DEPT:NUM     C     002
008      JOB:CODE     C     002
009      SALARY       N     008     002
010      ACT:STAT     L     001
** TOTAL **             00059
```

The structure displayed on your APC looks similar to the sample display screen, though the exact contents may differ depending on the file selected. The example shows the structure of EMPLOYEE.DBF.

The structure display includes the name of the file and the data field characteristics entered when the file was created. It also includes the number of records in the file, the date of last update, and the length of each record. The length is the sum of the individual field lengths plus one. dBASE II uses the extra character to identify deleted records in a file. "PRIMARY USE DATA BASE" means that this is the one data base in USE.

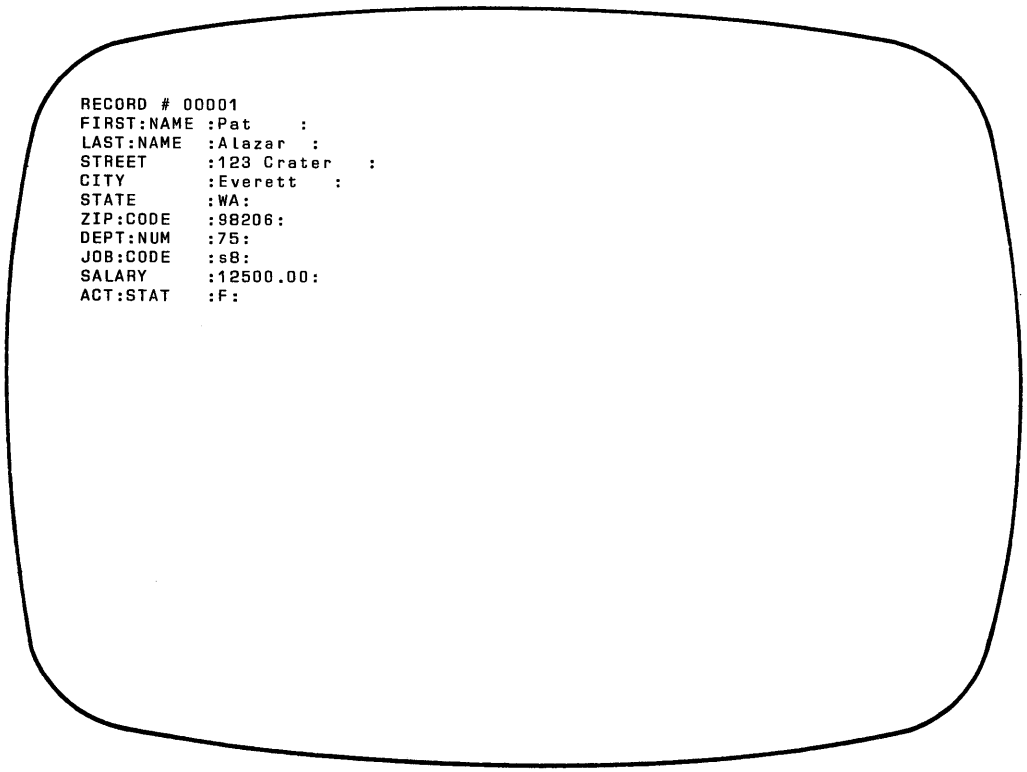
FULL-SCREEN EDITING

USE and LIST the JOBDET data base file. Notice that when you LIST the file, each record is preceded by a five-digit *sequence number*. Check the data for accuracy. If you have made any errors, you can EDIT them now. If you haven't made any errors, take a few minutes to try the EDIT functions anyway. You can make errors and correct them as many times as you like.

From the display, you know which records need correction. Enter the EDIT command with the sequence number of one record to be changed. The following command allows you to edit RECORD # 00001.

edit 1

The EDIT command erases the screen and displays the entire record, as in the example that follows. You can use the dBASE II editing commands to modify some, none, or all of the data fields, to move forward and backward between records in the file, and to save your changes or exit from the edit mode without modifying the data.



Moving the Cursor

You have a choice of ways to move the cursor in edit mode. The APC keyboard includes keys that are clearly labelled to indicate the operation performed.

The four cursor movement keys on the numeric keypad move the cursor in the direction shown by the arrows: up, down, left, and right.

BACKSPACE is located in the upper right corner of the alphanumeric keyboard. It moves the cursor back one space within a field, and back one field from the first character of a field.

TAB moves the cursor ahead one field. To tab back one field, press **SHIFT TAB.**

dBASE II also includes alternative edit commands that use combinations of the control key (**CTRL**) and a letter to move the cursor. **CTRL X** means hold down the key on the left side of the keyboard labelled **CTRL** while you press the letter **X**.

The letter used in combination with **CTRL - E, S,** and **D** - are located near each other on the left side of the keyboard. These three keys form a triangle shape. Their positions relative to each other indicate the direction in which they move the cursor.

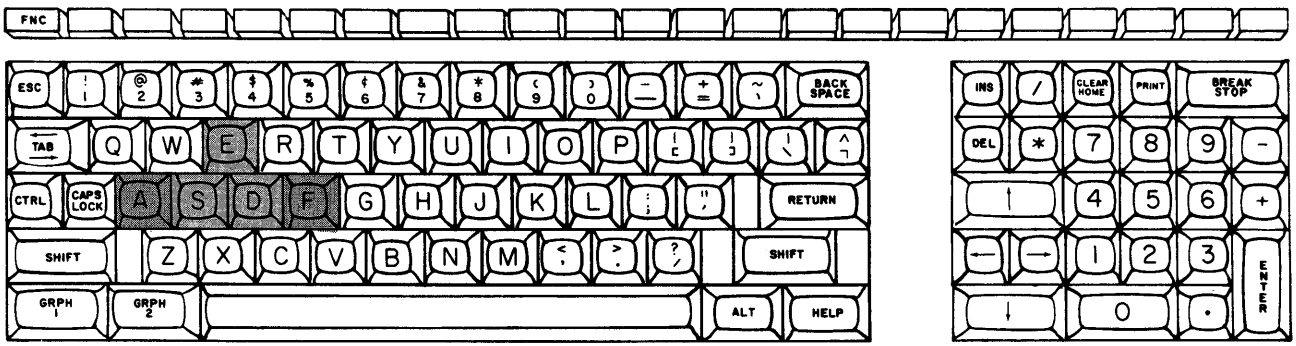


Figure 5-2 Cursor Movement Keys for Full-Screen Editing

CTRL D moves the cursor one character to the right.
CTRL S moves the cursor one character to the left.
CTRL E moves the cursor up to the previous field.

Two other letter keys - **A** and **F** - also work in combination with **CTRL** to control cursor movement. They are located to the left and right of the triangle shape formed

by the other letter keys. They extend the width of the triangle and, in editing, extend the scope of the cursor's movement.

CTRL A moves the cursor back to the previous field.
CTRL F moves the cursor ahead to the next field.

Overtyping

Position the cursor on the first character to be changed and type over it. In the *overtype mode,* you can replace data simply by typing over existing characters.

Inserting Characters

The **INS** key puts the system in *insert mode.* This allows you to insert characters into existing entries without destroying the data that already exists. To insert data, position the cursor on the character that is to immediately follow the insertion and press **INS**. The word "INSERT" appears at the top of the screen next to the record number. Then, type the insertion. The character under the cursor and all characters to the right of the cursor are shifted one position to the right for each inserted character. Press **INS** again to return to the overtype mode.

For example, to correct the name "HARMAN" to "HARDIMAN" you would follow this procedure:

1. Position the cursor on the M in HARMAN.
2. Press **INS** to enter the insert mode.
3. Type the two letters DI.
4. Press **INS** to enter the overtype mode.

CTRL V works like **INS** to toggle between the insert and overtype modes.

Deleting Characters

The **DEL** key deletes the character under the cursor and shifts the characters following the cursor one space to the left.

CTRL G and **CTRL X** work exactly like **DEL** to delete the character under the cursor.

Exiting the Edit Mode

When you have edited the records that need modification, press:

CTRL W to save changes to the current record and return to the dBASE II prompt;

CTRL Q to abort changes to the current record and return to the dBASE II prompt.

NOTE

The system automatically saves the changes to the current record and returns to the dot prompt when you press **RETURN** from the last field of the last record in a data base.

Table 5-5 lists all of the dBASE II edit mode commands. Some of these were described in this chapter. Others are explained later in the manual. Several command options are available for most edit functions. It is easiest to use the labelled keys.

Table 5-5 Full Screen Edit Commands - EDIT

COMMAND	ACTION
TAB ↓ **CTRL F** **CTRL J**	Moves cursor down to the next field
SHIFT TAB ↑ **CTRL A** **CTRL E** **CTRL K**	Moves cursor up to the previous field *
→ **CTRL D**	Moves cursor ahead one character
← **CTRL S**	Moves cursor back one character within a field, and back one field from the first character of a field *
DEL **CTRL G** **CTRL X**	Deletes character under the cursor
CTRL C	Saves changes and advances to the next record
CTRL R	Saves changes and backs up to the previous record
CTRL W	Saves changes to current record and returns to dBASE II command mode
CTRL Q	Aborts changes in current record and returns to dBASE II command mode
CTRL Y	Erases field
CTRL U	Toggles record deletion mark on/off
INS **CTRL V**	Toggles insert/overtype modes
PRINT **CTRL P**	Toggles printer on/off

*When the cursor is positioned on the first character of the first field in a record, this comand moves the cursor back to the first character in the first field of the previous record.

ERROR CORRECTION DIALOG

As you enter commands, the system scans them for accuracy. If dBASE II finds something it doesn't recognize, it displays a message on the screen immediately. You have the choice of reentering the entire command or fixing the errors. Using the *correction dialog* to fix the error can save a lot of typing.

To see how this feature works, deliberately make a mistake. Enter:

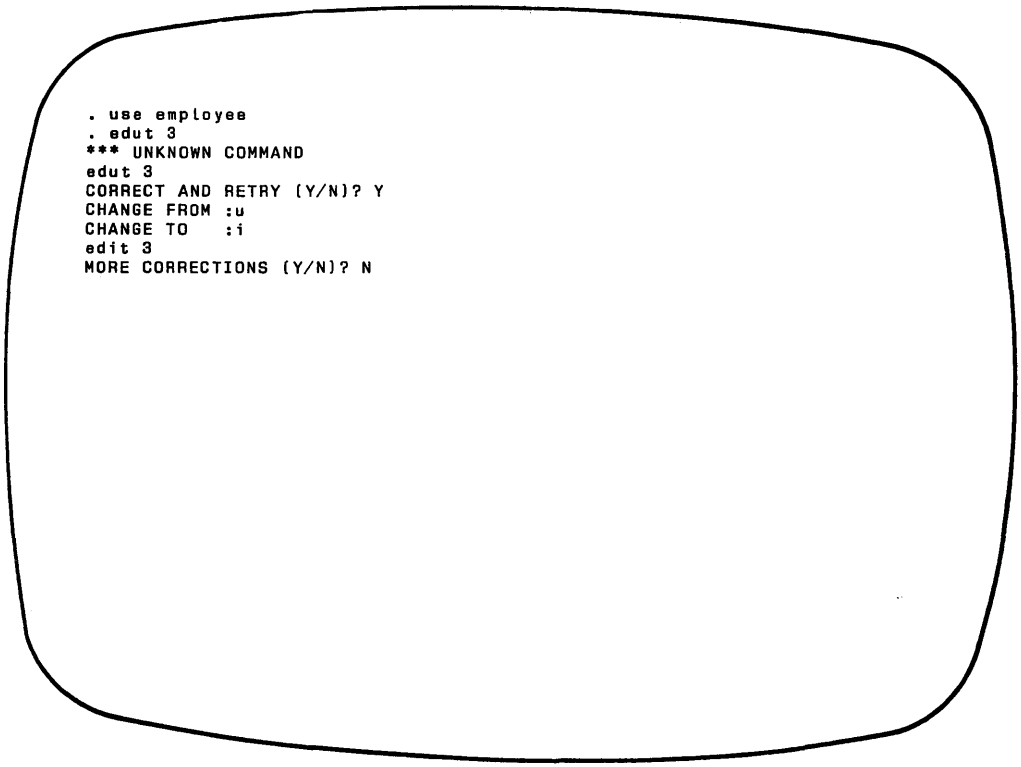
```
use employee
edut 3
```

dBASE II does its best to identify the error to you. When it detects an error that it cannot describe explicitly, it assumes that the error is a syntax error. The system displays the erroneous line back to you. If there is a syntax error, a question mark appears above the beginning of the phrase that caused the confusion.

Since "edit" was misspelled, the system displays the following prompt.

```
***UNKNOWN COMMAND
   edut 3
   CORRECT AND RETRY (Y/N)?
```

If you press **Y**, the dialog continues. If you press **N**, the system returns to the command mode. If you press any other key, the system displays"?" because it does not recognize the response. Press **Y**. The correction dialog begins, as shown in the next screen.



```
. use employee
. edut 3
*** UNKNOWN COMMAND
edut 3
CORRECT AND RETRY (Y/N)? Y
CHANGE FROM :u
CHANGE TO   :i
edit 3
MORE CORRECTIONS (Y/N)? N
```

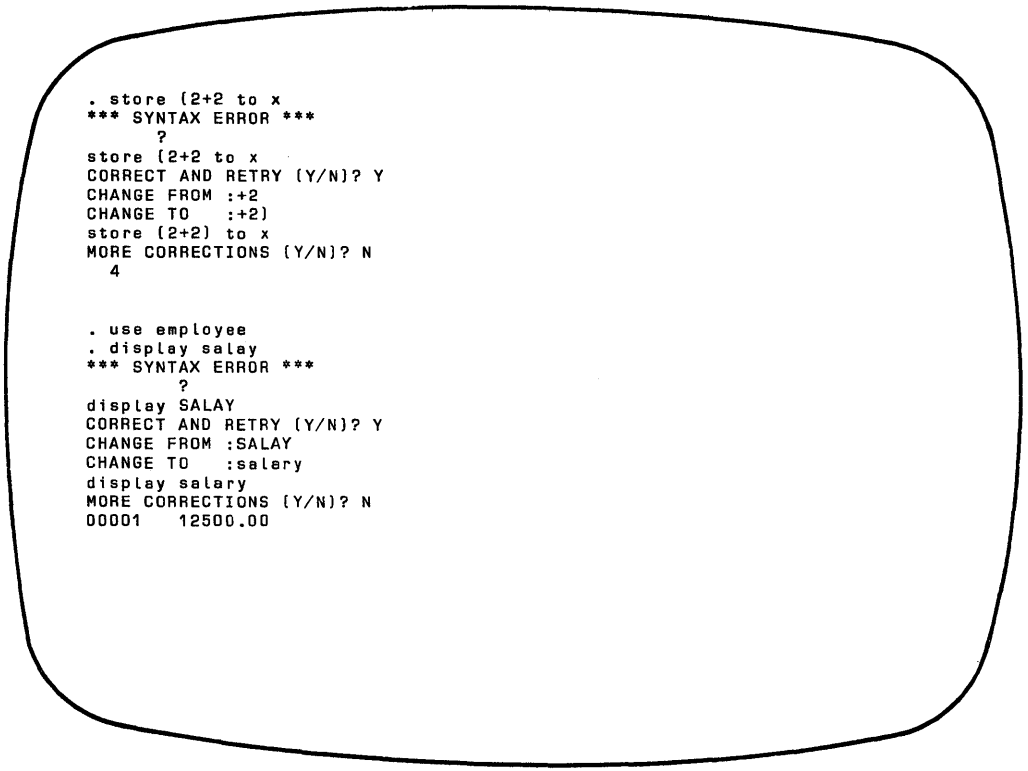
In response to "CHANGE FROM:" enter just enough of the wrong part of the command so that it is unambiguous. In the example, only one letter ("u") is wrong and it is the only "u" that was typed. So enter that letter.

In response to "CHANGE TO:", enter only the replacement for the material to be changed. Again, it is just one letter, "i".

To signal that all corrections have been made, press **N** in response to the question "MORE CORRECTIONS (Y/N)?".

Different errors can be corrected and you can change the same error as many times as necessary until the entire command is correct. When the command is correct, dBASE II allows the dialog to end and executes the command. You can exit from the correction dialog at any time by pressing **ESC**. Pressing **RETURN** in response to the prompt "CHANGE TO:" also exits from the dialog.

Two more examples of the error correction dialog are shown in the next screen. The first demonstrates how the system identifies syntax errors with the question mark.



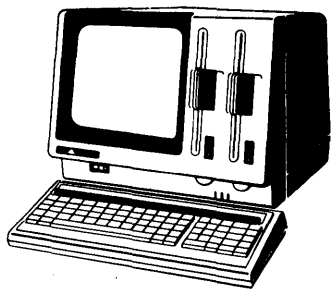
```
. store (2+2 to x
*** SYNTAX ERROR ***
        ?
store (2+2 to x
CORRECT AND RETRY (Y/N)? Y
CHANGE FROM :+2
CHANGE TO   :+2)
store (2+2) to x
MORE CORRECTIONS (Y/N)? N
  4


. use employee
. display salay
*** SYNTAX ERROR ***
        ?
display SALAY
CORRECT AND RETRY (Y/N)? Y
CHANGE FROM :SALAY
CHANGE TO   :salary
display salary
MORE CORRECTIONS (Y/N)? N
00001    12500.00
```

The second example demonstrates that in response to the prompt "CHANGE FROM:", you must type the error exactly as it is repeated back by the system, which may not be the way you entered it. dBASE II displays erroneous file names and field names in uppercase letters. It then compares the CHANGE FROM string you enter to the command line it displayed back to you. If the data do not match exactly, the system repeats the prompt "CHANGE FROM:".

In these examples, only a few characters were changed. You will find the error correction dialog a useful feature when entering long commands.

Chapter 6

Using The Data Base Selectively



You create data bases in order to get information from them. With dBASE II, you don't need to specify exactly what that information is until you want it. You don't have to write computer programs to access or report on file contents. Instead, you can get answers to questions as they arise. dBASE II allows you to query the data base in two ways: record location and record contents.

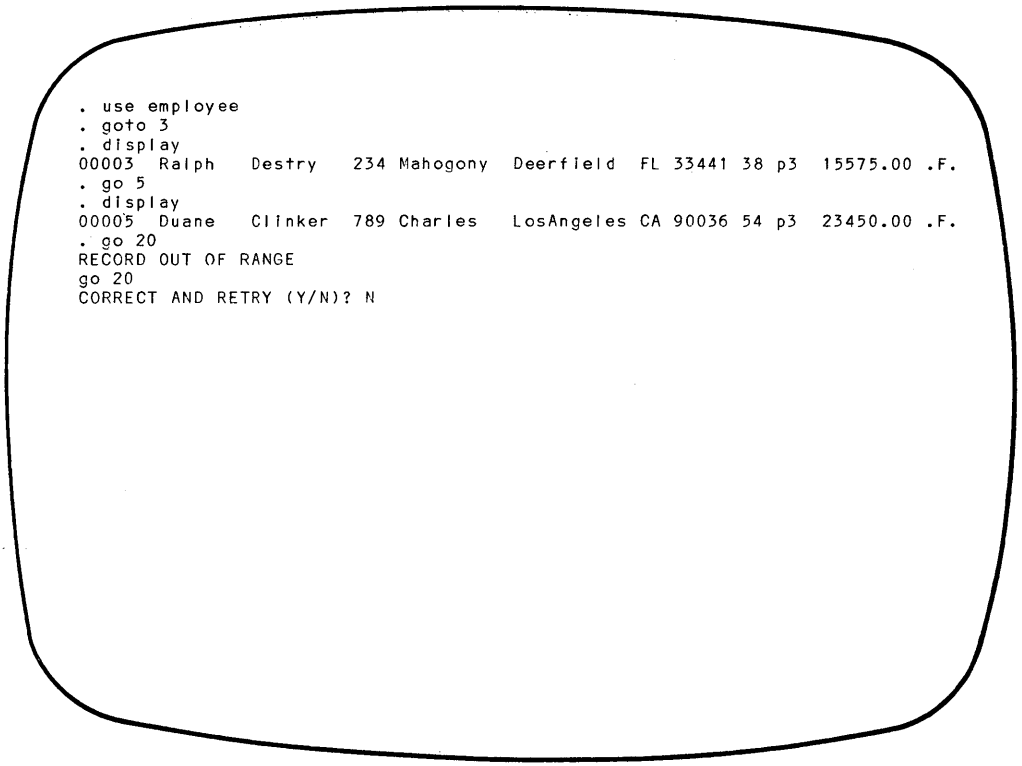
POSITIONING YOURSELF IN THE DATA BASE

Sometimes it is most convenient to operate on records based on their locations in the file. Assume that EMPLOYEE records are added to the file in the order in which people are hired. The first person hired is entered as the first record; the tenth record belongs to the tenth person hired, and so on. If you need to know who was hired first, sixty-third, or anywhere in between, you could access the appropriate record by its location.

GOTO

The GOTO command moves you from record to record quickly and easily. Enter the commands listed below at the dot prompt. When you finish, the screen should look like the display that follows.

```
use employee
goto 3
display
go 5
display
go 20
```



```
. use employee
. goto 3
. display
00003  Ralph   Destry   234 Mahogony  Deerfield  FL 33441 38 p3  15575.00 .F.
. go 5
. display
00005  Duane   Clinker  789 Charles   LosAngeles CA 90036 54 p3  23450.00 .F.
. go 20
RECORD OUT OF RANGE
go 20
CORRECT AND RETRY (Y/N)? N
```

Notice that when the specified record number is greater than the number of records in the data base file, the message "RECORD OUT OF RANGE" appears and you are prompted to reenter a number through the error correction dialog.

The GOTO command has five formats.

GOTO TOP
GOTO BOTTOM
GOTO RECORD <n>
GOTO <n>
<n>

NOTE

Pointed brackets ("<" and ">") in command format descriptions enclose text that you must replace with your own information. For example, in the GOTO command you must replace <n> with a valid record number.

- GO and GOTO are equally acceptable and may be used interchangeably.
- TOP and BOTTOM are *keywords*, words that have special meaning to dBASE II. They must be spelled correctly. Like commands, keywords can be abbreviated to the first four characters. TOP moves to the first record in the data base. BOTTOM moves to the last record.
- GOTO RECORD <n>, GOTO <n>, or even <n> alone (where "n" is a record number) moves to the indicated record. The three commands produce identical results.

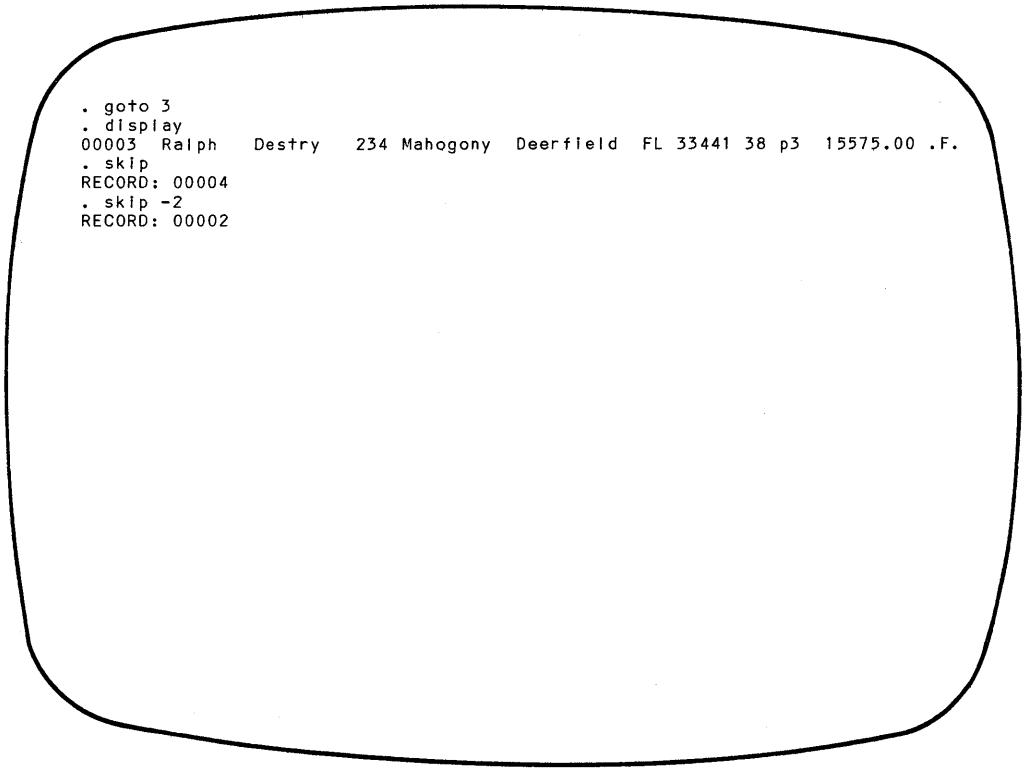
GOTO BOTTOM can be used to find the number of records in sequential files. Since record sequence numbers are assigned as records are added to a data base file, you can find the last record's number by issuing the GOTO BOTTOM command.

GOTO by itself may not seem like a very useful command. However, as you will see later, it is very useful in combination with other commands and in programs.

Skipping Around

You can move forward and backward from your current position with SKIP. Enter these commands to see how SKIP works.

```
goto 3
display
skip
skip-2
```



The SKIP command operates according to the following rules.

- SKIP by itself moves forward to the next record.
- SKIP + or - <n> moves forward or backward "n" records.
- When <n> is omitted, it is assumed to be 1.

The system always responds with the record number at its new location. Do not try to skip beyond the end of a file. If you do, the message "RECORD OUT OF RANGE" appears on the display screen instead of a record number. The system remains positioned at the last valid record number it used, however the field it calls "RECORD NUMBER" holds the out of range value.

Current Record Pointer

Although you may sometimes need to know which record in the file you are positioned on, dBASE II always needs to keep track of its current position. So the system automatically stores the current record number in its memory in a *variable* named #. A variable is simply a name assigned to a piece of data. A field name is another example of a variable. The *value* of a variable is its contents (for example, the number of the current record or a particular employee's name). The values a variable takes on may change, but you can always refer to the variable by name to find out its contents.

In the previous examples, DISPLAY was used to show the current record and to demonstrate the effects of the GOTO and SKIP commands. This displayed the entire record. If all you need to know is the current record number, displaying the value of the record pointer, #, accomplishes the same thing.

? Command

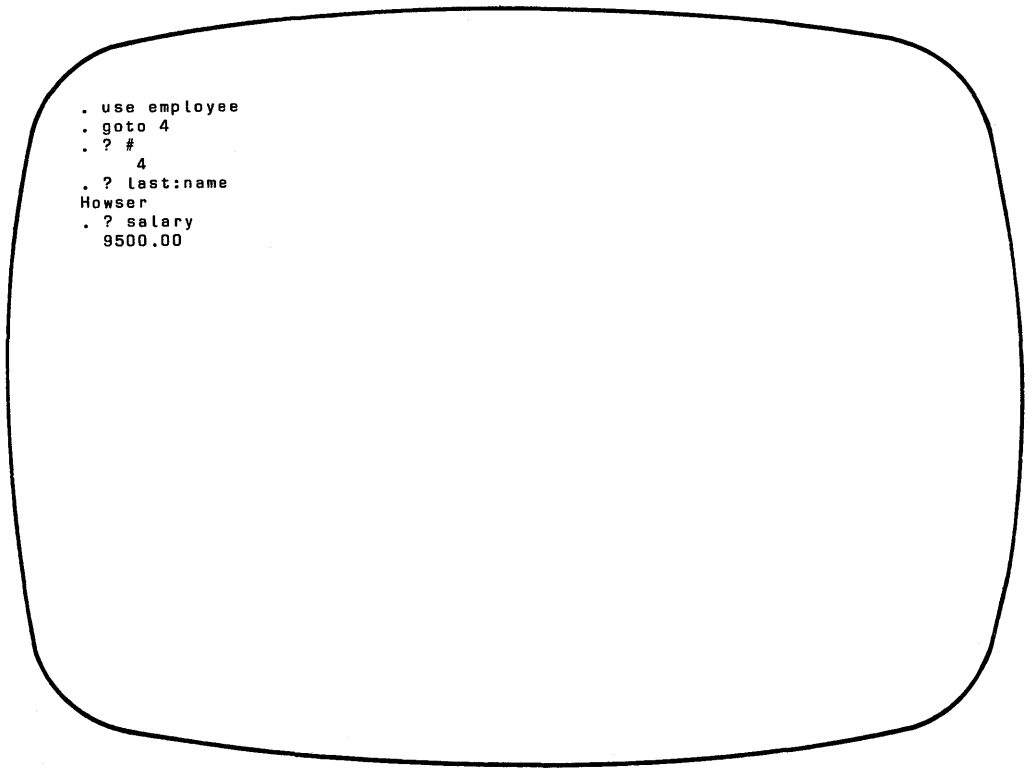
The ? command is a specialized form of the DISPLAY command. ? means "What is ..." to dBASE II. To ask "What is the value of the record pointer?" enter:

```
? #
```

The system responds with the number of the record at which it is positioned. If you then want to see the entire record, enter:

```
display
```

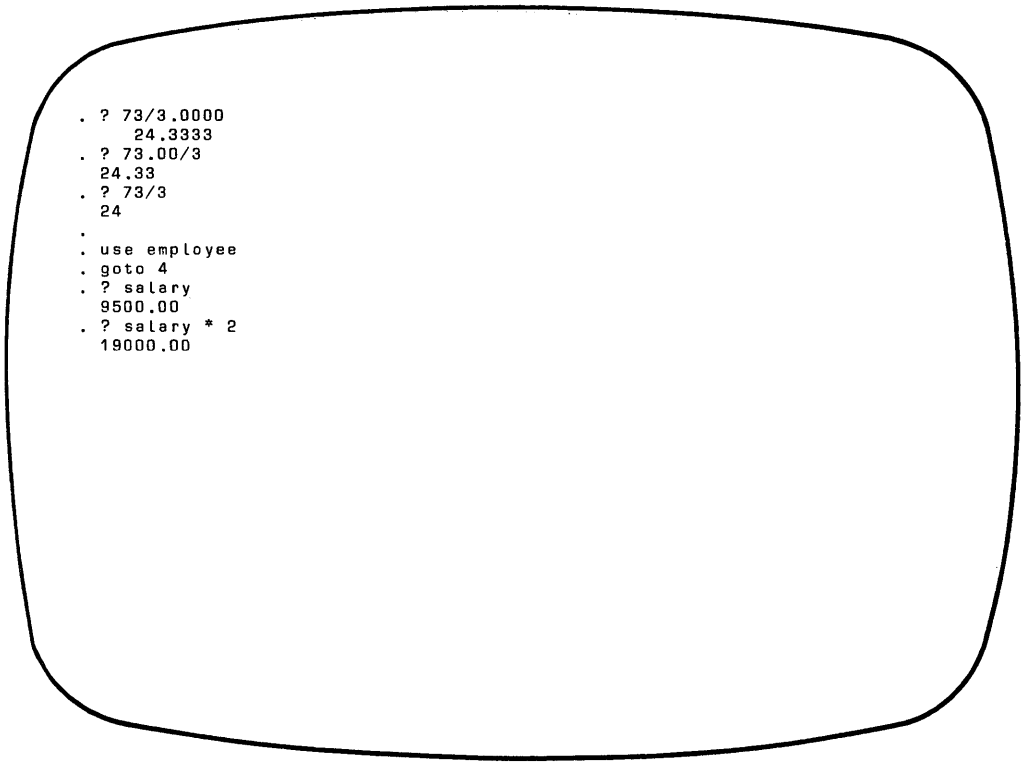
? can be used to display the value of any variable. Try using it to display the values of variable names in the data base currently in USE.



Calculator Mode

The ? command also lets you use dBASE II like a calculator. Simply enter the question mark followed by a space and the quantity or mathematical expression to be evaluated. dBASE II returns the answer on the next line of the screen.

Test the command with some calculations of your own or try the examples in the display that follows. Notice that the answers to mathematical operations are accurate to the maximum number of decimal places in the numbers entered.



? can evaluate expressions containing both variables and numbers.

?? Command

The command ?? performs the same functions as ?, but places the answer on the same line of the display screen as the command.

Try the ?? command in calculator mode and also to display the value of a variable. You will see that the answer is displayed starting at the left margin of the screen. The answer overlays as much of the command line as necessary. The rest of the command remains displayed on the screen.

dBASE II SYNTAX

You use commands to communicate with the dBASE II programs to access and manipulate your data. So far, most of the commands used have consisted of a verb (USE, LIST, DISPLAY) and a noun (the name of a file or field). To use dBASE II to its fullest, you can tailor these commands with modifiers and refine their operation.

Every computer program is written in a language that the computer can understand. You have probably heard of COBOL, FORTRAN, and BASIC. These are all computer programming languages. They are easy for computers to understand, but they are not so easy for humans to learn. dBASE II, on the other hand, can understand and accept commands that are also fairly easy for people to understand and remember. Its language consists of structures with which you're already familiar. The language resembles English and, like any language, it has a syntax (set of grammatical rules) to follow. This section describes the syntax in general terms. Once you understand the terminology and typographic conventions, you can take full advantage of the dBASE II commands.

Command Format

In general, the format of a dBASE II command is:

```
VERB SCOPE FIELD CONDITION
```

A *verb* is an action word. CREATE, USE, LIST, REPORT, and ERASE are all examples of verbs. They cause a specific action to be taken. The SCOPE, FIELD, and CONDITION *phrases* that follow the verb are equivalent to adverbs. They more fully describe the action.

- SCOPE identifies how much of the file the command covers.
- FIELD identifies specific parts of records to be used. The way each field is used depends on the command.
- CONDITION states the selection criteria for records.
- Some phrases require special formats. FOR, NEXT, and WITH are examples of words that begin phrases. All of these example words are referred to as "keywords".

The dBASE II language includes rules that limit the acceptable combination of verbs and phrases. Each verb accepts only certain phrases. If the phrase is present, it must conform to the rules. When phrases are omitted, the system automatically supplies *default* values for them so that the command is complete.

RULES FOR ENTERING COMMANDS

dBASE II accepts commands that adhere to the following format.

- The verb must be the first non-blank character of the command line.
- Phrases follow the verb and can be in any order.
- Any number of blanks may be used to separate words and phrases.
- Verbs and keywords can be abbreviated to the first four (or more) characters. For example, DISPLAY STRUCTURE may be entered as DISP STRU or DISPL STRUCT, and so on. Just be sure the abbreviation is spelled correctly up to the point where it ends. All examples in this manual use complete words for clarity.
- Either uppercase or lowercase letters may be used to enter verbs, keywords, field names, memory variable names, or file names. dBASE II converts them to uppercase.
- All commands must contain fewer than 254 characters. This count includes blanks. If macros (see Chapter 11) are used, the length includes the expanded form of the macro. To extend a command line beyond the width of the display screen, use a semicolon (;) after the last character that fits on the line. Press **RETURN** immediately after the semicolon and continue the command on the next line.

SPECIAL SYMBOLS USED IN COMMAND FORMATS

Parts of the commands are optional and may be omitted when the command is used. A typical command format is shown below.

LIST [<scope>] [<field list>] [FOR <exp>]

Square brackets ([...]) are used in the command formats to show which phrases are optional. These are the phrases that modify the action of commands.

Uppercase type identifies keywords (for example, FOR) in phrases. Keywords must be entered whenever the phrase that contains them is used.

NOTE

Although it is possible to use keywords as file and/or field names, it is a good practice to avoid doing so. dBASE II may incorrectly interpret your use of them, causing errors and unpredictable results.

Pointed brackets (<...>) enclose words, phrases, and abbreviations that describe the data you enter. These symbols are defined in Appendix B. You must replace the entire symbol (the brackets and the words they enclose) with your own values.

Expanding your Control with Phrases

To see how verbs and phrases interact, enter the following series of commands:

```
use employee
list
goto top
list next 3
goto top
list next 3 last:name
goto top
list last:name for dept:num='89'
```

These commands demonstrate how phrases can restrict the action of LIST. The phrases are defined more fully in this section.

SCOPE

The scope phrase determines how much of the file is considered when the command is executed. Every command has a default for scope. See what happens when you enter these commands:

```
use employee
display
list
```

Without any modifying phrases, DISPLAY shows only one record (the current record) on the screen while LIST shows the whole file.

Both LIST and DISPLAY can be used to show the contents of a file. They both accept the same phrases. However, they differ in the default for SCOPE. For any command that uses this phrase, it can take on three values:

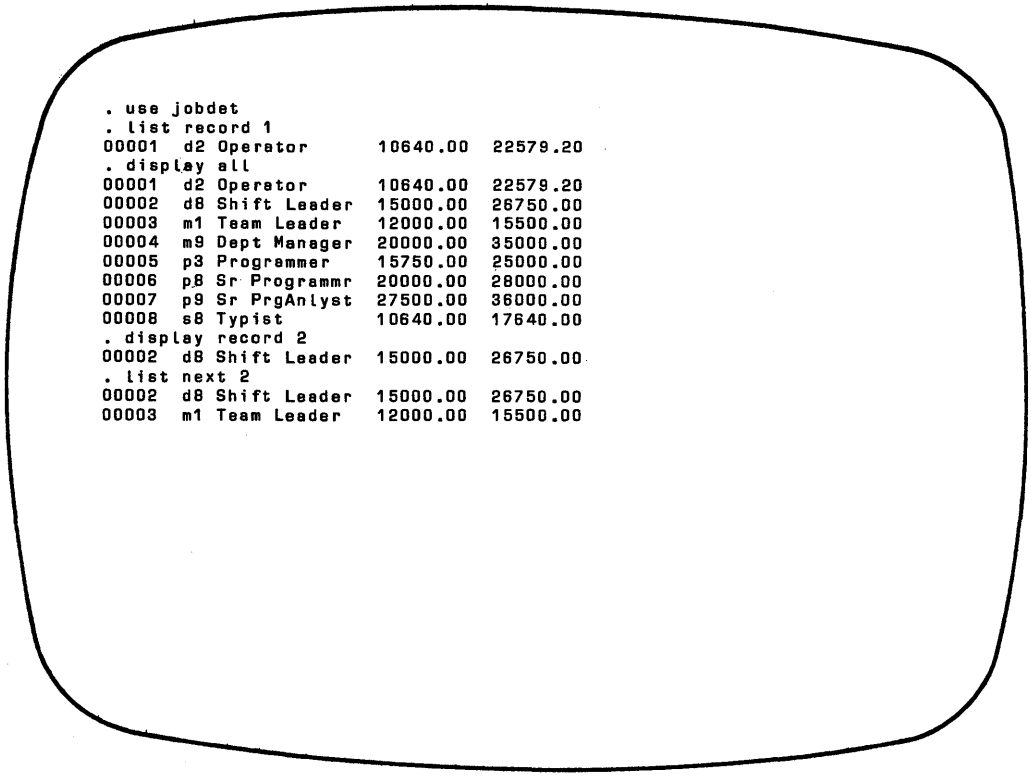
- ALL means all the records in the data base;
- NEXT <n> means the next "n" records, beginning with the current record;
- RECORD <n> means the record with the sequence number "n".

Try the <scope> phrase with both LIST and DISPLAY. The general form of the commands with this phrase is:

```
LIST [<scope>]
DISPLAY [<scope>]
```

Enter these commands. The screen appears like the display that follows.

```
use jobdet
list record 1
display all
display record 2
list next 2
```



```
. use jobdet
. list record 1
00001   d2 Operator       10640.00   22579.20
. display all
00001   d2 Operator       10640.00   22579.20
00002   d8 Shift Leader   15000.00   26750.00
00003   m1 Team Leader    12000.00   15500.00
00004   m9 Dept Manager   20000.00   35000.00
00005   p3 Programmer     15750.00   25000.00
00006   p8 Sr Programmr   20000.00   28000.00
00007   p9 Sr PrgAnlyst   27500.00   36000.00
00008   s8 Typist         10640.00   17640.00
. display record 2
00002   d8 Shift Leader   15000.00   26750.00
. list next 2
00002   d8 Shift Leader   15000.00   26750.00
00003   m1 Team Leader    12000.00   15500.00
```

LIST and DISPLAY can produce identical screen displays when the scope phrase is used. Table 6-1 describes the effect of the scope phrase on the LIST and DISPLAY commands.

Table 6-1 Effects of Scope Phrase With LIST and DISPLAY

LIST COMMAND	DISPLAY COMMAND	ACTION
LIST	DISPLAY ALL	Contents of entire file are displayed and current record pointer is set to the last record
LIST NEXT 1	DISPLAY	Current record is displayed
LIST RECORD <n>	DISPLAY RECORD <n>	Indicated record is displayed and current record pointer is reset

FIELDS

Each of your files will probably consist of many fields. Long records may be an efficient way to organize and store data, however you do not necessarily want to use all the fields at one time.

When you display records to answer a particular question, the screen should be as clear and uncluttered as possible. One way to accomplish this is to use a FIELD phrase, when it is allowed, so that what you see is just what you need.

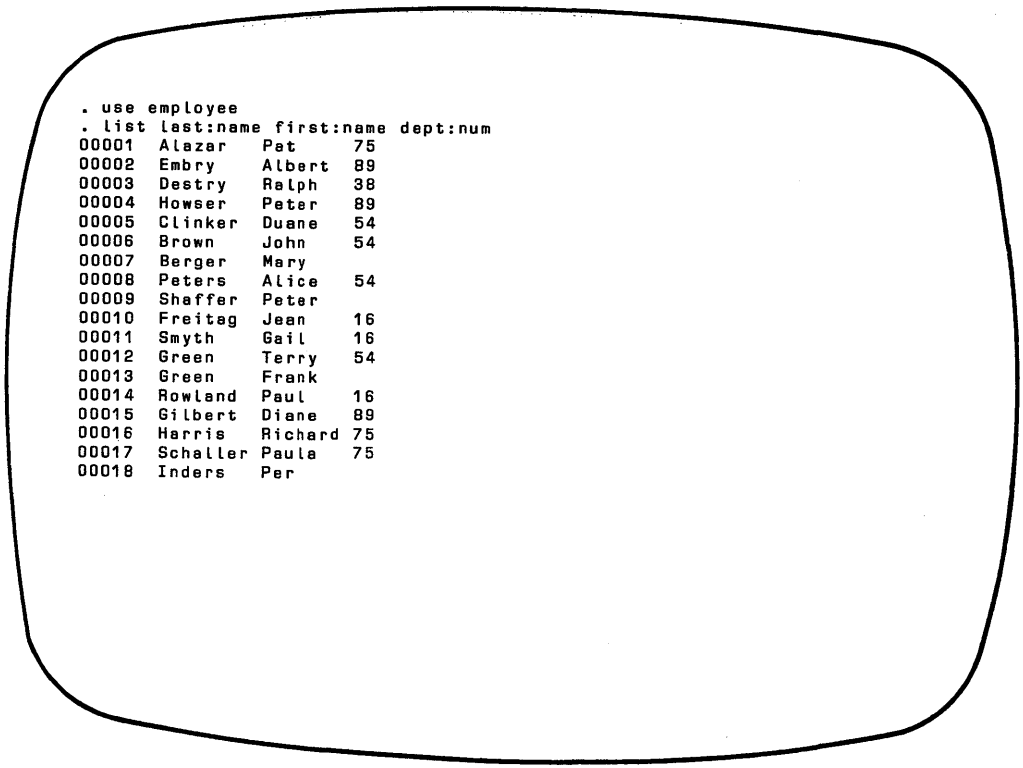
LIST and DISPLAY can use a field phrase to limit the display to only those fields included in the <field list>. No keywords are used with this phrase.

LIST [<scope>] [<field list>]
DISPLAY [<scope>] [<field list>]

For a list of all employees and where in the company they work, you need only the names and associated department numbers for every record in the EMPLOYEE file. LIST can be expanded with a phrase so that only those fields are shown. Enter:

```
use employee
list last:name first:name dept:num
```

The system displays the record number and those three fields for each record.

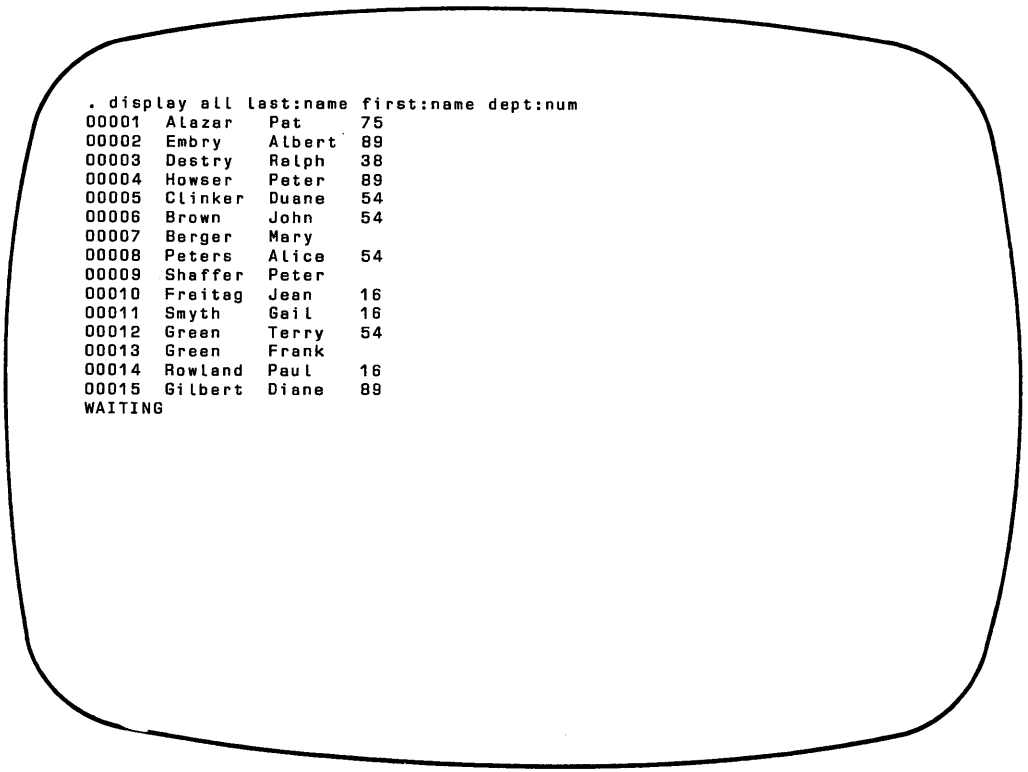


```
. use employee
. list last:name first:name dept:num
00001  Alazar   Pat      75
00002  Embry    Albert   89
00003  Destry   Ralph    38
00004  Howser   Peter    89
00005  Clinker  Duane    54
00006  Brown    John     54
00007  Berger   Mary
00008  Peters   Alice    54
00009  Shaffer  Peter
00010  Freitag  Jean     16
00011  Smyth    Gail     16
00012  Green    Terry    54
00013  Green    Frank
00014  Rowland  Paul     16
00015  Gilbert  Diane    89
00016  Harris   Richard  75
00017  Schaller Paula    75
00018  Inders   Per
```

You can use DISPLAY to accomplish the same thing as the previous LIST command by also specifying the scope. Enter:

display all last:name first:name dept:num

As you can see in the display that follows, the program responds by displaying 15 records at a time and the message "WAITING". Signal that you are ready to continue with the display by pressing any key.



```
. display all last:name first:name dept:num
00001  Alazar    Pat      75
00002  Embry     Albert   89
00003  Destry    Ralph    38
00004  Howser    Peter    89
00005  Clinker   Duane    54
00006  Brown     John     54
00007  Berger    Mary
00008  Peters    Alice    54
00009  Shaffer   Peter
00010  Freitag   Jean     16
00011  Smyth     Gail     16
00012  Green     Terry    54
00013  Green     Frank
00014  Rowland   Paul     16
00015  Gilbert   Diane    89
WAITING
```

LIST and DISPLAY can produce the same output, but they present it in different ways. LIST scrolls through the display and stops only on your signal. Press **CTRL S** or **BREAK** to start and stop a LISTing. Press **ESC** to abort the LIST command. DISPLAY presents output in groups of 15 records. Press any key to continue with the next group of records.

NOTE

Pressing **ESC** aborts any dBASE II command when you are in command mode.

CONDITIONS

Conditional phrases are very useful for controlling and defining what the commands will do. Conditional phrases consist of *keywords* and *expressions* - constants and/or variables joined by operators. The following command uses an expression to restrict record selection to those employees meeting a certain condition:

```
list for salary = 9500
```

In that command, "for" is a keyword and "salary = 9500" is an expression made up of a *variable* (salary), a *constant* (9500), and an *operator* (=).

Keywords

dBASE II recognizes two keywords that identify conditional phrases: FOR and WHILE. They can be used interchangeably in all commands except LOCATE. The LOCATE command does not accept the keyword WHILE, however LOCATE does recognize conditional phrases introduced by FOR.

NOTE

For consistency, FOR is used throughout this manual as the keyword that identifies conditional phrases. You can use FOR or WHILE.

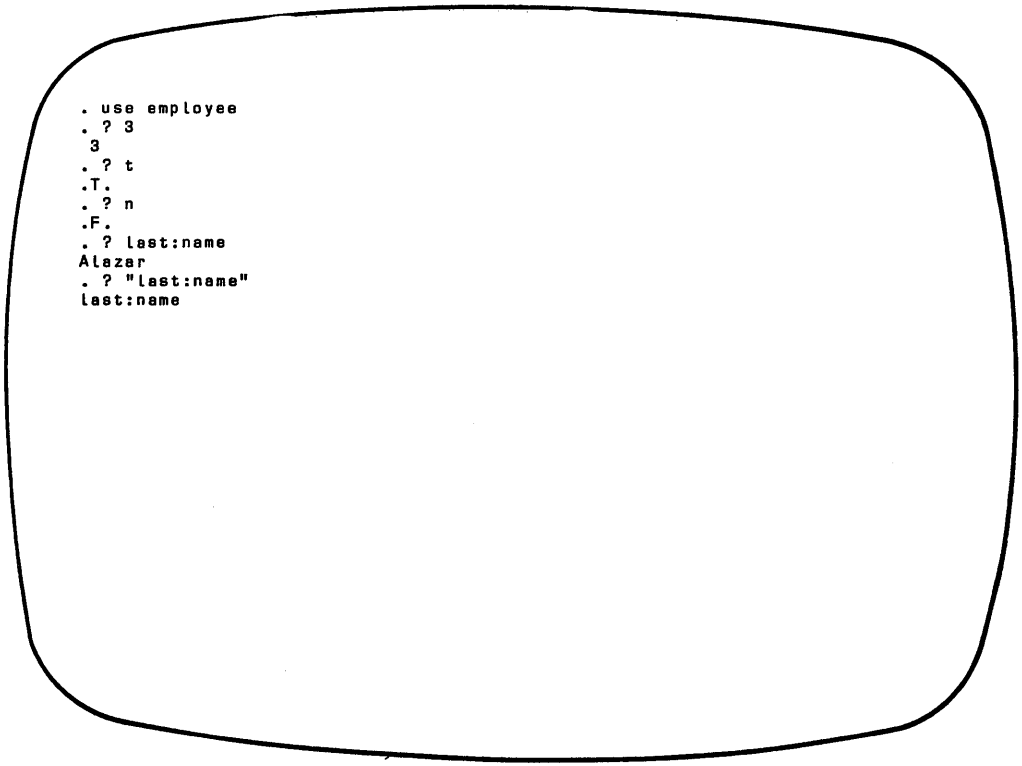
Variables

Variables are data items whose values can change. Frequently they are the names of data base fields, like LAST:NAME, DEPT:NUM, and SALARY. They can also be *system* variables like record number (#), and memory variables (see Chapter 12).

Constants

Constants are data items that do not change, no matter where they appear in a data base or within the system. They are also called *literals* because they are exactly what they say. Constants can be one of three types: numeric, logical, or character.

- *Numeric constants* are numerals such as 3.102, 175, and 42.
- *Logical constants* are the values ".T." and ".F.".
- *Character constants* are called *strings*. They may be composed of numbers, letters, and symbols and are always enclosed within single or double quotes or brackets. To see how the computer handles constants, look at the following screen.



```
. use employee
. ? 3
 3
. ? t
.T.
. ? n
.F.
. ? last:name
Alazar
. ? "last:name"
last:name
```

There is no confusion about the first three commands. When you enter "**?3**" ("What is 3?") the system responds "3". Three is always 3. The symbols T, t, Y, and y are logical constants that mean "True", so "**?t**" gives the answer ".T.". Similarly, F, f, N, and n mean "False". But look at what happened with the last two commands. The first time the program was asked "What is last:name?" it displayed the current value of the variable called LAST:NAME. The second time, it displayed the string constant "last:name". Character string constants must be enclosed within quotes to distinguish them from variables.

Operators

Literals and constants must be combined with *operators* to form expressions. *Operators* are symbols that tell dBASE II what manipulations to perform on the data. dBASE II recognizes four kinds of operators. Some of them will be familiar to you; others may take a bit of practice to understand how they work.

Arithmetic operators should be the most familiar. They generate arithmetic results.

Table 6-2 Arithmetic Operators

OPERATOR	MEANING
( )	Parentheses for grouping
*	Multiplication
/	Division
+	Addition
-	Subtraction

The arithmetic operators are evaluated in a sequence of precedence. The order is: parentheses; multiplication and division; addition and subtraction. When the operators have equal precedence, they are evaluated from left to right. The following examples demonstrate the effect of parentheses on normal arithmetic operations.

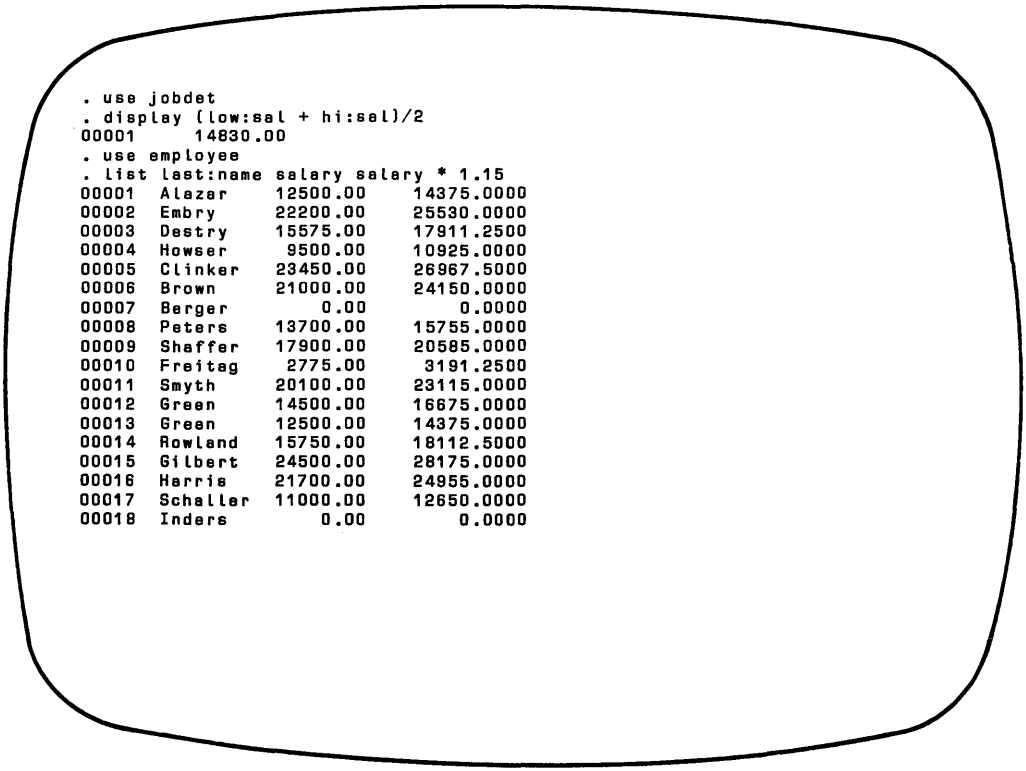
```
17/33*72+8     = 45.09     (divide, multiply, add)
17/(33*72+8)   =  0.00713  (multiply, add, divide)
17/33*(72+8)   = 41.21     (add, divide, multiply)
```

NOTE

In dBASE II, the results of arithmetic operations are carried out to the greatest number of decimal places in the operands. Therefore, the previous examples would yield the following answers in dBASE II:

```
17/33*72+8     = 45
17/(33*72+8)   =  0
17/33*(72+8)   = 41
```

Two arithmetic operations are demonstrated in the screen that follows. The first calculates the midpoint of one salary range. The second example lists the current salary for each employee and the salary with a 15% increase.



```
. use jobdet
. display (low:sal + hi:sal)/2
00001     14830.00
. use employee
. list last:name salary salary * 1.15
00001  Alazar     12500.00     14375.0000
00002  Embry      22200.00     25530.0000
00003  Destry     15575.00     17911.2500
00004  Howser      9500.00     10925.0000
00005  Clinker    23450.00     26967.5000
00006  Brown      21000.00     24150.0000
00007  Berger         0.00         0.0000
00008  Peters     13700.00     15755.0000
00009  Shaffer    17900.00     20585.0000
00010  Freitag     2775.00      3191.2500
00011  Smyth      20100.00     23115.0000
00012  Green      14500.00     16675.0000
00013  Green      12500.00     14375.0000
00014  Rowland    15750.00     18112.5000
00015  Gilbert    24500.00     28175.0000
00016  Harris     21700.00     24955.0000
00017  Schaller   11000.00     12650.0000
00018  Inders         0.00         0.0000
```

Relational Operators make comparisons, then generate logical results. If the expression is true, the command is performed. If the expression is false, the command is not performed.

Table 6-3 Relational Operators

OPERATOR	MEANING
<	Less than
>	Greater than
=	Equal to
<=	Less than or equal to
>=	Greater than or equal to
<> or #	Not equal to

NOTE

The symbols <=, >=, and <> must be entered as two characters with no intervening spaces.

Relational operators can compare variables, constants, and expressions. The data types on both sides of the operator must be the same. For character strings, the data are compared from left to right for the length of the second argument (the string to the right of the relational operator). For numeric strings, an arithmetic comparison is made (for example, 7.00 equals 7). Try the following commands.

```
use employee
list for zip:code<='70000'
list for job:code<>'p'
list for last:name='Green'
list for salary>23000
```

```
. use employee
. list for zip:code <= '70000'
00003  Ralph   Destry   234 Mahogony  Deerfield  FL 33441 38 p3  15575.00 .F.
00004  Peter   Howser   678 Dusty Rd  Chicago    IL 60631 89 s8   9500.00 .F.
00006  John    Brown    456 Minnow Pl Burlington MA 01730 54 p3  21000.00 .F.
00007  Mary    Berger   10 Kearney Rd Needham    MA 02194    p8      0.00 .F.
00008  Alice   Peters   676 Wacker Dr Chicago    IL 60606 54 s8  13700.00 .F.
00009  Peter   Shaffer  43 Clinton Av Montclair  NJ 07042    d8  17900.00 .F.
00011  Gail    Smyth    817 5th St.   Ambler     PA 19147 16 p4  20100.00 .F.
00013  Frank   Green    441 Spicer Av Tampa      FL 33622    s8  12500.00 .F.
00016  Richard Harris   101 Enders Dr Syracuse   NY 13211 75 d8  21700.00 .F.
. list for job:code <> 'p'
00001  Pat     Alazar   123 Crater    Everett    WA 98206 75 s8  12500.00 .F.
00004  Peter   Howser   678 Dusty Rd  Chicago    IL 60631 89 s8   9500.00 .F.
00008  Alice   Peters   676 Wacker Dr Chicago    IL 60606 54 s8  13700.00 .F.
00009  Peter   Shaffer  43 Clinton Av Montclair  NJ 07042    d8  17900.00 .F.
00012  Terry   Green    567 Doheny Dr Hollywood  CA 90044 54 m1  14500.00 .F.
00013  Frank   Green    441 Spicer Av Tampa      FL 33622    s8  12500.00 .F.
00016  Richard Harris   101 Enders Dr Syracuse   NY 13211 75 d8  21700.00 .F.
00017  Paula   Schaller 721 Spring St Everett    WA 98206 75 s8  11000.00 .F.
00018  Per     Inders   321 Sawtelle  Tuscon     AZ 85702            0.00 .F.
. list for last:name = 'Green'
00012  Terry   Green    567 Doheny Dr Hollywood  CA 90044 54 m1  14500.00 .F.
00013  Frank   Green    441 Spicer Av Tampa      FL 33622    s8  12500.00 .F.
. list for salary > 23000
00005  Duane   Clinker  789 Charles   LosAngeles CA 90036 54 p3  23450.00 .F.
00015  Diane   Gilbert  280 Cactus Wy Las Cruces NM 88001 89 p3  24500.00 .F.
```

Notice that quotes or square brackets must enclose character data.

Logical Operators are used to test the values in logical fields and to form complex expressions. They generate logical results (true or false). The logical operators are listed in Table 6-4 in the order of precedence within an expression.

Table 6-4 Logical Operators

OPERATOR	MEANING
( )	parentheses for grouping
.NOT.	boolean not
.AND.	boolean and
.OR.	boolean or
$	substring logical operator

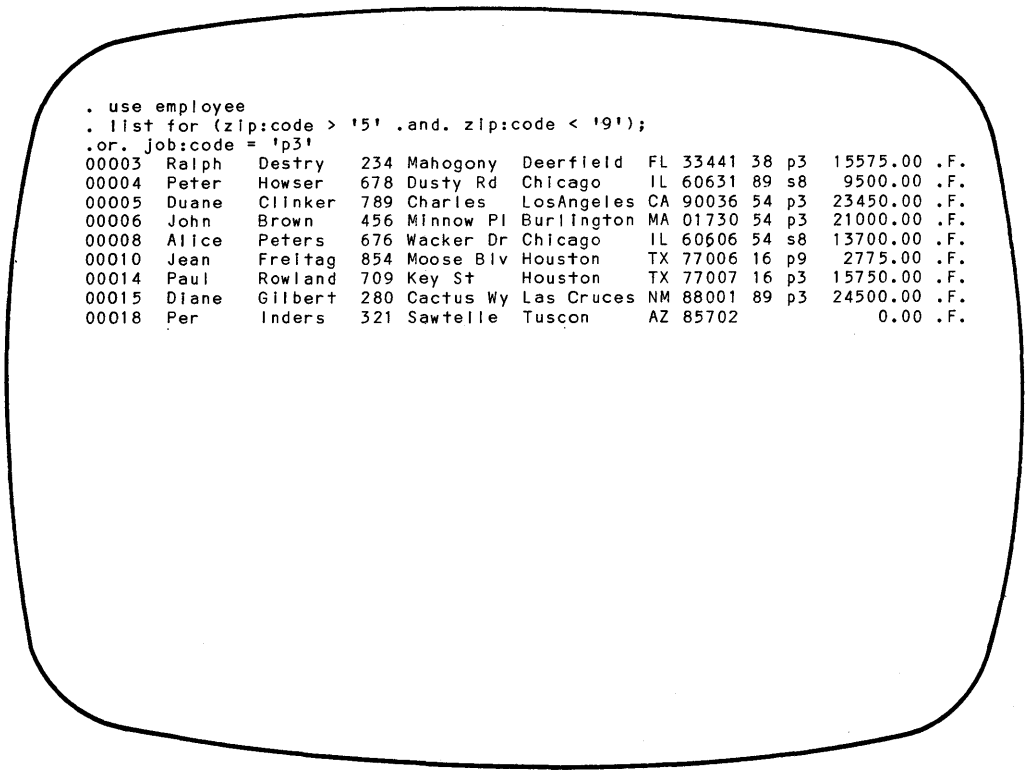
Logical Operators are used to test the values in logical fields. For instance, to find the records for all employees with the ACT:STAT field value equal to F, enter:

```
list for .not.act:stat
```

The system responds with all employees' records because the value of ACT:STAT is ".F." in all records.

You can use the logical operators to combine arithmetic and relational expressions into complex expressions like the example that follows. Since the command does not fit on one line, the semicolon is used to mark the end of the first line and the comand is continued on the next line.

```
use employee
list for (zip:code>'5' .and.zip:code<'9');
.or.job:code='p3'
```



```
. use employee
. list for (zip:code > '5' .and. zip:code < '9');
.or. job:code = 'p3'
00003  Ralph   Destry    234 Mahogony  Deerfield  FL 33441 38 p3  15575.00 .F.
00004  Peter   Howser    678 Dusty Rd  Chicago    IL 60631 89 s8   9500.00 .F.
00005  Duane   Clinker   789 Charles   LosAngeles CA 90036 54 p3  23450.00 .F.
00006  John    Brown     456 Minnow Pl Burlington MA 01730 54 p3  21000.00 .F.
00008  Alice   Peters    676 Wacker Dr Chicago    IL 60606 54 s8  13700.00 .F.
00010  Jean    Freitag   854 Moose Blv Houston    TX 77006 16 p9   2775.00 .F.
00014  Paul    Rowland   709 Key St    Houston    TX 77007 16 p3  15750.00 .F.
00015  Diane   Gilbert   280 Cactus Wy Las Cruces NM 88001 89 p3  24500.00 .F.
00018  Per     Inders    321 Sawtelle  Tuscon     AZ 85702            0.00 .F.
```

The system evaluates complex expressions with logical operators in two stages. First, each relational or arithmetic expression is evaluated for the data values in a record. (In the example, there are three such expressions.) Then, the system performs the logical tests (in this case .and., .or.) before deciding whether to display the record.

Parentheses are used in logical operations just as they are used in arithmetic expressions. In the example, the expressions within the parentheses are evaluated first. If both expressions within the parentheses evaluate ".T.", the first logical test evaluates ".T.". In that case, the next logical test does not have to be made. If either expression within the parentheses was not true, then the entire expression within the parentheses would be false and the second logical expression would be evaluated. Either the expression within parentheses *or* the other expression must be true for a record to be displayed. If the second logical operator were ".and." (as in the command LIST FOR (ZIP:CODE > '5' .AND. ZIP:CODE < '9') .AND. JOB:-CODE = 'p3'), then *both* logical tests would have to be true for a record to be displayed. That would result in the display of two employees' records, Rowland and Gilbert.

The *Substring Logical Operator ($)* searches for a given string of characters within another string and returns a true or false value. The format is:

<substring> $ <string>

Either or both terms may be string variables or constants. To see how this works, enter the following:

```
use employee
display
? state $ "CALIFORNIA"
? state $ "WASHINGTON"
list for 'ee' $ name
list for 'CA' $ state
```

```
. use employee
. display
00001  Pat      Alazar    123 Crater     Everett     WA 98206 75 s8   12500.00 .T.
. ? state $ "CALIFORNIA"
.F.
. ? state $ "WASHINGTON"
.T.
. list for "ee" $ last:name
00012  Terry    Green     567 Doheny Dr  Hollywood   CA 90044 54 m1   14500.00 .F.
00013  Frank    Green     441 Spicer Av  Tampa       FL 33622    s8   12500.00 .F.
. list for 'CA' $ state
00002  Albert   Embry     345 Sage Ave   Palo Alto   CA 94303 89 p8   22200.00 .F.
00005  Duane    Clinker   789 Charles    LosAngeles  CA 90036 54 p3   23450.00 .F.
00012  Terry    Green     567 Doheny Dr  Hollywood   CA 90044 54 m1   14500.00 .F.
```

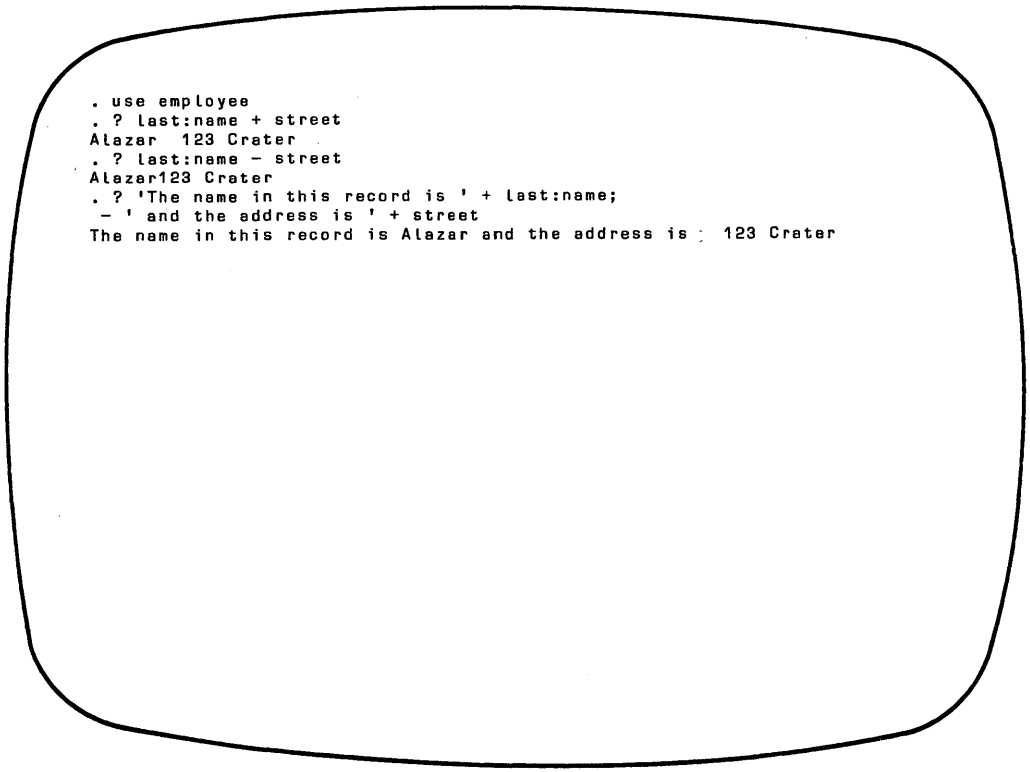
String Operators generate string results.

Table 6-5 String Operators

OPERATOR	MEANING
+	String concatenation
-	String concatenation, move blanks

Concatenation is a computer buzzword. It means that one character string is stuck on to the end of another one. Enter the following series of commands:

```
use employee
? last:name + street
? last:name - street
? 'The name in this record is '+name;
  -' and the address is '+ street
```



```
. use employee
. ? last:name + street
Alazar  123 Crater
. ? last:name - street
Alazar123 Crater
. ? 'The name in this record is ' + last:name;
 - ' and the address is ' + street
The name in this record is Alazar and the address is  123 Crater
```

The + and - operators both join two strings. The plus sign joins the strings exactly as they are found. The minus sign moves the trailing blanks in a string to the end of the concatenated string. The blanks are not eliminated, but for many purposes this is enough since they do not show up between the strings that are joined.

FINDING RECORDS BY THEIR CONTENTS

The previous sections of this chapter have demonstrated how to use LIST and DISPLAY with conditional phrases to locate data records that meet specified criteria. In addition to LIST and DISPLAY, a third command that is useful for retrieving specific records is LOCATE. It searches the data base records in the USE file for the first record whose data fields allow the conditional expression to be true. When the expression is satisfied, the record number is displayed. The complete form of the LOCATE command is:

LOCATE [<scope>] [FOR <exp>]

The verb, like any verb or keyword, may be abbreviated to four or more letters.

The scope phrase is optional; it defaults to ALL. To search the entire data base, omit the scope phrase. To search part of a file, use "NEXT <n>" for the scope. The search starts at the current record and looks at the next "n" records. If this would move the pointer past the end of the file, LOCATE examines each record from the pointer position to the end of the file.

The FOR phrase specifies the conditions which must be true if the record is to be selected. The keyword "FOR" is required. The expression may be simple (AGE > 21) or complex (AGE >= 25 .and. JOB:CODE = "p9" .or. NAME='GOU') . Notice that character data must be enclosed in single or double quotes.

To test how LOCATE works, enter the following commands:

```
use employee
locate for dept:num = '89'
```

If a record is found that meets the conditions in the expression, dBASE II returns the message "RECORD n", where "n" is the record number. You can display or edit the record once it is located.

If a record cannot be found, the message "END OF FILE ENCOUNTERED" is displayed and the current record pointer is positioned on the last record in the file. If the NEXT phrase is used and no record can be found within the scope of the phrase, the message "END OF LOCATE SCOPE" is displayed and the current record pointer indicates the last record scanned.

There may be more than one record that meets the conditions specified in the LOCATE command. CONTINUE tells the system to go on with the search.

Other dBASE II commands may be issued between LOCATE and CONTINUE. If they are, all <exp> phrases in the LOCATE command and the following commands are limited to 128 characters each instead of the usual 254 characters.

Try the LOCATE - CONTINUE commands. The display that follows shows how the commands operate.

```
. use employee
. locate next 3 for dept:num = '99'
END OF LOCATE SCOPE
. display
00003  Ralph    Destry    234 Mahogony  Deerfield  FL 33441 38 p3  15575.00 .F.
. go top
. locate for dept:num = '99'
END OF FILE ENCOUNTERED
. display
00017  Per      Inders    321 Sawtelle  Tuscon     AZ 85702           0.00 .F.
. go top
. locate for dept:num = '89'
RECORD: 00002
. display
00002  Albert   Embry     345 Sage Ave  Palo Alto  CA 94303 89 p8  22200.00 .F.
. continue
RECORD: 00004
. display
00004  Peter    Howser    678 Dusty Rd  Chicago    IL 60631 89 s8   9500.00 .F.
. continue
RECORD: 00015
. display
00015  Diane    Gilbert   280 Cactus Wy Las Cruces NM 88001 89 p3  24500.00 .F.
. continue
END OF FILE ENCOUNTERED
```

REFINING SOME FAMILIAR COMMANDS

Now that you are familiar with the command symbols and their definitions, take a few minutes to look over the commands described below. This list summarizes the command formats discussed so far in this manual.

? <exp list>

?? <exp list>

APPEND

CREATE [<file>]

GOTO <numeric exp>

DISPLAY [<scope>] [<field list>] [FOR <exp>] [OFF]
LIST [<scope>] [<field list>] [FOR <exp>] [OFF]

Some commands can take special optional phrases that are unique to them. [OFF] is one such phrase. When it is used, the record sequence numbers are not DISPLAYed or LISTed. USE a file and try the following commands to test the effects of OFF.

```
list
list off
```

DISPLAY FILES [ON <disk drive>] [LIKE *.<type>]

LIST FILES [ON <disk drive>] [LIKE *.<type>]

This form of the LIST and DISPLAY commands presents the names of files on a diskette. When no optional phrase is used, all files on the default drive with the ".DBF" extension are listed.

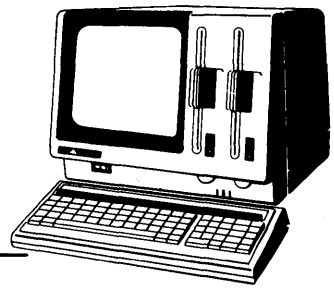
The ON phrase specifies a disk drive other than the default.

The LIKE phrase allows any file type to be displayed. Replace <type> with the extension (NDX, TXT, FRM, and so on). So far, you've used only DBF and FRM files, but you'll find this command useful later after you've created other types of files.

DISPLAY STRUCTURE

LIST STRUCTURE

SKIP <numeric exp>



Chapter 7

Modifying A Data Base

It is inevitable that data bases will change. You will have to respond to changing governmental regulations, modifications in company policies, and your own realization that you can make your data bases better as you gain practice with them. The personnel data base will change as employees transfer between departments. Their salaries will change. New employees will be hired and others will retire and resign. The company will have changing information needs. When a new benefit is offered -leaves of absence, for instance - the personnel department will need more categories for employee status than just TRUE or FALSE (Active or Not Active). To track salary history for each employee, the data base would need an entirely new record type that used some information already available on the EMPLOYEE records.

These examples illustrate the three kinds of changes that can be made to data bases. First, the contents of fields can change. Second, entire records can be added and deleted. Third, the data base structure can be altered. This chapter discusses how to change records and fields. Chapter 8 describes commands and procedures for changing the data base structure.

ADDING NEW RECORDS

Append

You are already familiar with how the APPEND command works to add records to the end of a file. This command can be used with both empty files and files that contain some data records.

APPEND, with no modifiers, puts the system in data entry mode. You can add records to the end of the file one after another until you signal to return to command mode.

dBASE II has a number of commands that control how it interacts with the APC. They are called SET commands and they do things like turn the printer on and off, set margins, and enable and disable the bell. You can change these parameters back

and forth at will. One such feature you may find useful when APPENDing records is the capability of carrying data forward from one record to the next. SET CARRY ON does this. It is especially useful if successive records have a lot of common data. Records are automatically filled with data and can be edited as necessary.

Use the following procedure to carry data forward when adding records to the end of a file.

1. At the dot prompt, enter the following commands.

   use <file>
   set carry on
   append

   The screen is erased and the system is placed in data entry mode. The record structure is displayed and the fields are filled with their values from the last record in the file.

2. Edit the record as necessary. Overtype or insert data, or press **RETURN** for each field.

   If the last field in the record is modified, the system automatically saves the record and prepares to accept another record by displaying the next sequential record number. The data field values remain intact.

   If the last field in the record is not modified, press **CTRL C** to continue adding records or **CTRL W** to exit from the data entry mode.

3. Repeat Step 2 until all additions are complete. Then press **CTRL W** when the cursor is on the first character of the first field of the record display to return to control mode.

4. At the dot prompt, enter the following command to reset the CARRY feature.

   set carry off

APPEND BLANK, a second form of the command, adds a single, default-filled record to the file that is in USE. All character fields are filled with spaces, numeric fields are zero-filled, and logical fields are assigned the value ".F.". The system remains in command mode. To supply field values, EDIT the record or use the REPLACE command, described later in this chapter.

The following screen demonstrates how APPEND BLANK works using a sample file consisting of three field types: character, numeric, and logical.

```
. use testapnd
. display structure
STRUCTURE FOR FILE:  TESTAPND.DBF
NUMBER OF RECORDS:   00003
DATE OF LAST UPDATE: 11/04/82
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH    DEC
001      ALPHA        C     005
002      NUMER        N     003
003      LOG          L     001
** TOTAL **              00010
. list
00001  Alpha  111 .T.
00002  Beta   222 .F.
00003  Delta  333 .T.
. append blank
. list
00001  Alpha  111 .T.
00002  Beta   222 .F.
00003  Delta  333 .T.
00004            0 .F.
```

Insert

Records can be added to data base files at specific locations (to keep them alphabetical, for example) with INSERT. This command assigns the record sequence number, accepts input of the data fields, and renumbers all subsequent records in the file. The full description of the INSERT command is:

INSERT [BEFORE] [BLANK]

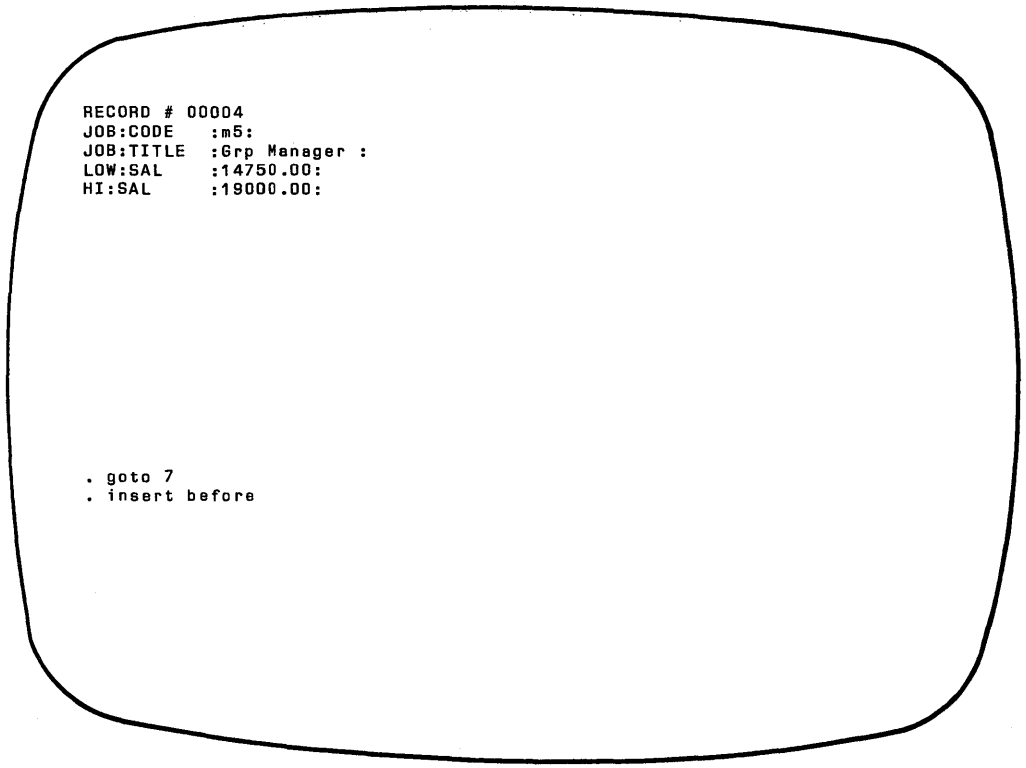
To INSERT a record at a particular location, use a command such as GOTO to position the current record pointer at the record that will immediately precede or follow the addition. Then use INSERT to format and add the record.

- INSERT alone places the new record just after the current record. INSERT BEFORE adds a record just before the current record. In either case, the system is placed in data entry mode and you are prompted for data in the same way as with CREATE and APPEND. Only one record is added at a time. The system returns to command mode after the last field of the record is filled with data.
- INSERT BLANK places a default-filled record in the data base file. The system remains in command mode.

If the SET CARRY ON command is in effect, the data fields are filled with the values from the record indicated by the current record pointer immediately prior to the INSERT command.

The sequence of commands displayed in the following screens inserts records into the JOB DETAILS file so that it is still in order by job code. Enter the commands into your system to update the data base file.

```
. use jobdet
. list
00001   d2 Operator        9500.00   20160.00
00002   d8 Shift Leader   15000.00   26750.00
00003   m1 Team Leader    12000.00   15500.00
00004   m9 Dept Manager   20000.00   35000.00
00005   p3 Programmer     15750.00   25000.00
00006   p8 Sr Programmr   20000.00   28000.00
00007   p9 Sr PrgAnlyst   27500.00   36000.00
00008   s8 Typist          9500.00   15750.00
. go to 3
. insert
```


RECORD # 00004
JOB:CODE :m5:
JOB:TITLE :Grp Manager :
LOW:SAL :14750.00:
HI:SAL :19000.00:
. goto 7
. insert before

```
RECORD # 00007
JOB:CODE    :p4:
JOB:TITLE   :Prog/Analyst:
LOW:SAL     :17000.00:
HI:SAL      :26500.00:



. list
00001   d2 Operator         9500.00   20160.00
00002   d8 Shift Leader    15000.00   26750.00
00003   m1 Team Leader     12000.00   15500.00
00004   m5 Grp Manager     14750.00   19000.00
00005   m9 Dept Manager    20000.00   35000.00
00006   p3 Programmer      15750.00   25000.00
00007   p4 Prog/Analyst    17000.00   26500.00
00008   p8 Sr Programmr    20000.00   28000.00
00009   p9 Sr PrgAnlyst    27500.00   36000.00
00010   s8 Typist          9500.00    15750.00
```

NOTE

INSERTs into a large non-indexed database take a long time to complete and should be avoided whenever possible. When you have many inserts, it is more efficient to APPEND the new records and then SORT the file than to INSERT each record at its proper location.

CLEANING UP A DATA BASE

Deleting Records

Removing records from a data base is a two-step process. First, identify the records to be deleted using either of the methods outlined below. This *logically* deletes the records but does not actually remove them from the data base, which is helpful in preventing catastrophic losses of data. You have a chance to recover your data before *physically* removing it from the data base. Records marked for deletion appear on LISTs and DISPLAYs of the data base. However, dBASE II bypasses these records in most other operations.

DELETING FROM THE COMMAND MODE

DELETE alone deletes the current record. It places an asterisk, called the *deletion mark*, in the first character position in the record. This is the extra character that dBASE II adds to the record length in the record structure. It tells the system to bypass the record when processing the file. The complete form of the DELETE command is:

DELETE [<scope>] [FOR <exp>]

- The default for <scope> is NEXT 1 (the current record). To delete a record other than the current record, use the <scope> phrase: ALL, RECORD <n>, or NEXT <n>.
- To make the deletions conditional, expand the command with the FOR phrase.
- When both the scope and a conditional phrase are used, the system deletes all records within the scope for which the expression is true.

To see how this command works, enter these commands.

```
use employee
delete record 5
delete next 3
list
```

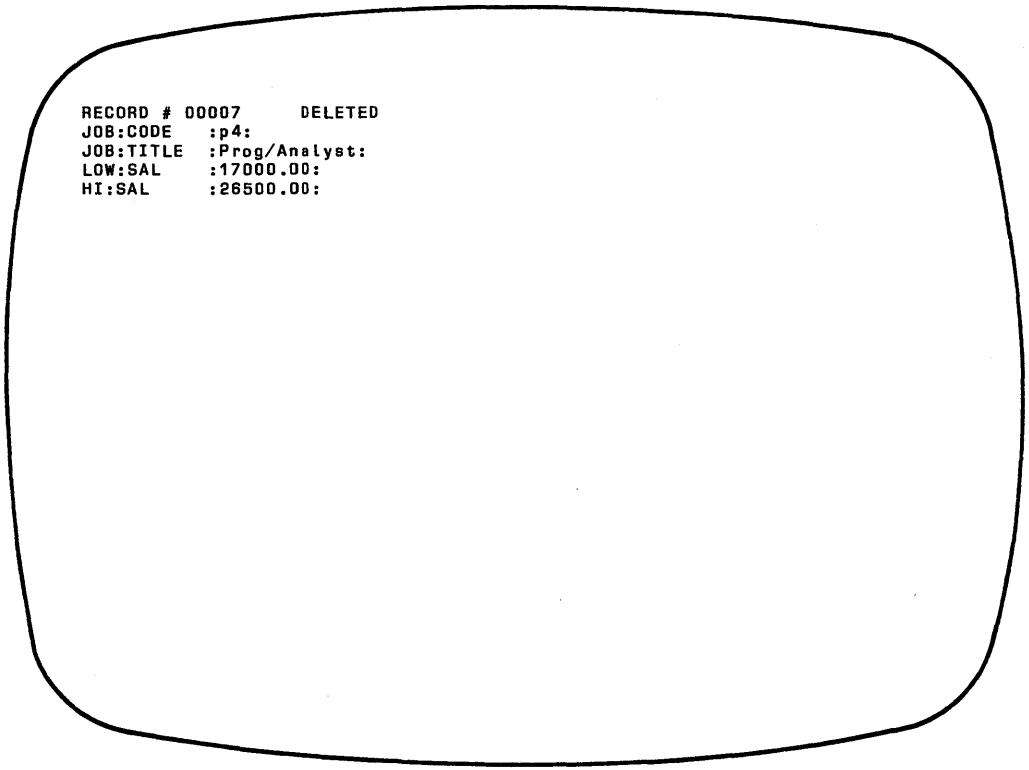
```
. use employee
. delete record 5
00001 DELETION(S)
. delete next 3
00002 DELETION(S)
. list
00001  Pat      Alazar   123 Crater     Everett     WA 98206 75 s8  12500.00 .F.
00002  Albert   Embry    345 Sage Ave   Palo Alto   CA 94303 89 p8  22200.00 .F.
00003  Ralph    Destry   234 Mahogony   Deerfield   FL 33441 38 p3  15575.00 .F.
00004  Peter    Howser   678 Dusty Rd   Chicago     IL 60631 89 s8   9500.00 .F.
00005 *Duane    Clinker  789 Charles    LosAngeles  CA 90036 54 p3  23450.00 .F.
00006 *John     Brown    456 Minnow Pl  Burlington  MA 01730 54 p3  21000.00 .F.
00007 *Mary     Berger   10 Kearney Rd  Needham     MA 02194    p8      0.00 .F.
00008  Alice    Peters   676 Wacker Dr  Chicago     IL 60606 54 s8  13700.00 .F.
00009  Peter    Shaffer  43 Clinton Av  Montclair   NJ 07042    d8  17900.00 .F.
00010  Jean     Freitag  854 Moose Blv  Houston     TX 77006 16 p9   2775.00 .F.
00011  Gail     Smyth    817 Sth St.    Ambler      PA 19147 16 p4  20100.00 .F.
00012  Terry    Green    567 Doheny Dr  Hollywood   CA 90044 54 m1  14500.00 .F.
00013  Frank    Green    441 Spicer Av  Tampa       FL 33622    s8  12500.00 .F.
00014  Paul     Rowland  709 Key St     Houston     TX 77007 16 p3  15750.00 .F.
00015  Diane    Gilbert  280 Cactus Wy  Las Cruces  NM 88001 89 p3  24500.00 .F.
00016  Richard  Harris   101 Enders Dr  Syracuse    NY 13211 75 d8  21700.00 .F.
00017  Paula    Schaller 721 Spring St  Everett     WA 98206 75 s8  11000.00 .F.
00018  Per      Inders   321 Sawtelle   Tuscon      AZ 85702            0.00 .F.
```

The first DELETE command moves the current record pointer to the fifth record in the file and deletes that record. The second DELETE command begins at the location of the current record pointer. It deletes the current record and the next two records (records 5, 6, and 7), and updates the current record pointer as it deletes the records.

DELETING FROM THE EDIT MODE

You can also delete records in edit mode. This is especially useful when you are moving sequentially through the records in a data base file making both changes and deletions. Deletions in edit mode are made by pressing **CTRL U.**

To try this, select any record in the USE file, enter the EDIT command, and press **CTRL U** when the record is displayed.



The word "DELETED" appears on the first line of the display. Press **CTRL U** again. The word "DELETED" disappears and the recors is restored.

The first step in deleting records only marks them for deletion. They are still *physically* on the file. They can be LISTed, DISPLAYed, and COUNTed. But they are *logically* deleted and are ignored by dBASE II in most other processing.

Restoring Records to the Data Base

Records marked for deletion can be recovered. In edit mode, **CTRL U** toggles the deletion mark on and off. In command mode, RECALL restores records.

RECALL [<scope>] [FOR <exp>]

RECALL operates exactly the opposite of DELETE. It removes the deletion mark from records. Scope and condition are optional. Scope defaults to the current

record. If a conditional expression is used, it does not have to be the same one that was used to mark the records for deletion. Enter:

```
list
recall for last:name = "B"
list
```

```
. recall for last:name = 'B'
00002 RECALL[S]
. list
00001  Pat      Alazar   123 Crater    Everett     WA 98206 75 s8  12500.00 .F.
00002  Albert   Embry    345 Sage Ave  Palo Alto   CA 94303 89 p8  22200.00 .F.
00003  Ralph    Destry   234 Mahogony  Deerfield   FL 33441 38 p3  15575.00 .F.
00004  Peter    Howser   678 Dusty Rd  Chicago     IL 60631 89 s8   9500.00 .F.
00005 *Duane,   Clinker  789 Charles   LosAngeles  CA 90036 54 p3  23450.00 .F.
00006  John     Brown    456 Minnow Pl Burlington  MA 01730 54 p3  21000.00 .F.
00007  Mary     Berger   10 Kearney Rd Needham     MA 02194    p8      0.00 .F.
00008  Alice    Peters   676 Wacker Dr Chicago     IL 60606 54 s8  13700.00 .F.
00009  Peter    Shaffer  43 Clinton Av Montclair   NJ 07042    d8  17900.00 .F.
00010  Jean     Freitag  854 Moose Blv Houston     TX 77006 16 p9   2775.00 .F.
00011  Gail     Smyth    817 Sth St.   Ambler      PA 19147 16 p4  20100.00 .F.
00012  Terry    Green    567 Doheny Dr Hollywood   CA 90044 54 m1  14500.00 .F.
00013  Frank    Green    441 Spicer Av Tampa       FL 33622    s8  12500.00 .F.
00014  Paul     Rowland  709 Key St    Houston     TX 77007 16 p3  15750.00 .F.
00015  Diane    Gilbert  280 Cactus Wy Las Cruces  NM 88001 89 p3  24500.00 .F.
00016  Richard  Harris   101 Enders Dr Syracuse    NY 13211 75 d8  21700.00 .F.
00017  Paula    Schaller 721 Spring St Everett     WA 98206 75 s8  11000.00 .F.
00018  Per      Inders   321 Sawtelle  Tuscon      AZ 85702        0.00 .F.
```

CAUTION

When there are two or more files in a data base, be careful as you add and delete records to keep the files in sync. In the personnel data base, the job code field is common to both files and is the way to join information from the two files. Be sure that you add employee records with valid job codes -codes that appear in the JOBDET.DBF file. Conversely, don't delete a record from the JOBDET.DBF file if there are still EMPLOYEE.DBF records in that category.

Permanently Removing Records from the Data Base

The second step in the deletion process permanently removes records from the file. It cleans up files to clarify displays and speed processing. PACK *physically* removes all records marked for deletion and tells you how many records are left in the data base file. It is a one-word command that operates on the data base file in USE. PACK is demonstrated below.

```
. pack
PACK COMPLETE, 00017 RECORDS COPIED
. list
00001  Pat      Alazar    123 Crater     Everett     WA 98206 75 s8  12500.00 .F.
00002  Albert   Embry     345 Sage Ave   Palo Alto   CA 94303 89 p8  22200.00 .F.
00003  Ralph    Destry    234 Mahogony   Deerfield   FL 33441 38 p3  15575.00 .F.
00004  Peter    Howser    678 Dusty Rd   Chicago     IL 60631 89 s8   9500.00 .F.
00005  John     Brown     456 Minnow Pl  Burlington  MA 01730 54 p3  21000.00 .F.
00006  Mary     Berger    10 Kearney Rd  Needham     MA 02194    p8      0.00 .F.
00007  Alice    Peters    676 Wacker Dr  Chicago     IL 60606 54 s8  13700.00 .F.
00008  Peter    Shaffer   43 Clinton Av  Montclair   NJ 07042    d8  17900.00 .F.
00009  Jean     Freitag   854 Moose Blv  Houston     TX 77006 16 p9   2775.00 .F.
00010  Gail     Smyth     817 Sth St.    Ambler      PA 19147 16 p4  20100.00 .F.
00011  Terry    Green     567 Doheny Dr  Hollywood   CA 90044 54 m1  14500.00 .F.
00012  Frank    Green     441 Spicer Av  Tampa       FL 33622    s8  12500.00 .F.
00013  Paul     Rowland   709 Key St     Houston     TX 77007 16 p3  15750.00 .F.
00014  Diane    Gilbert   280 Cactus Wy  Las Cruces  NM 88001 89 p3  24500.00 .F.
00015  Richard  Harris    101 Enders Dr  Syracuse    NY 13211 75 d8  21700.00 .F.
00016  Paula    Schaller  721 Spring St  Everett     WA 98206 75 s8  11000.00 .F.
00017  Per      Inders    321 Sawtelle   Tuscon      AZ 85702           0.00 .F.
```

WARNING

Once the PACK command has been issued, the records are gone forever.

SORTING THE FILE

SORT resequences the records in a file without changing the original file in any way. The records are sorted to another file in either ascending or descending order. In Chapter 3, you SORTed the PEOPLE file into alphabetical (ascending) order in a file named ABCORDER. The new file contains all the records that were in the PEOPLE file. The only difference between the two files is the order of the records.

SORTs are useful when a data base performs many functions and must be used in different orders for different purposes. For instance, if you want to keep the EMPLOYEE file sequenced by hiring order, but need to display or report information about employees in alphabetical order, you can SORT the EMPLOYEE file

into that order. This works well for small files, but is not efficient for large files. A better technique, indexing, is discussed in Chapter 9.

The most efficient way to add records to an ordered, sequential file is to APPEND new records to the end of the data base file and then SORT the file.

The complete form of the SORT commands is:

SORT ON <field> TO <file> [ASCENDING]
[DESCENDING]

The default for order is ASCENDING. SORT uses the ASCII collating sequence: numerals are "smallest", followed by uppercase letters and then lowercase letters. This means, for instance, that "SMITH" precedes "Smith" in a sorted file. (See Appendix F for the complete collating sequence.)

The file in USE remains in USE and is unaltered.

WARNING

DO NOT SORT A DATA BASE TO ITSELF. A power line "glitch" at the wrong moment could destroy your entire data base file. Instead, sort to another file and confirm the data.

Sorting on Multiple Keys

Although SORT operates on only one key field, you can organize a file by a series of key fields with multiple sorts. For instance, you might need to order employees alphabetically within job code for each department. Think of these fields as a hierarchy. In a listing or report, records could appear in the sequence shown in Figure 7-1.

DEPT.	JOB CODE	NAME
16	p3	Rowland, Paul
	p4	Smyth, Gail
	p9	Freitag, Jean
38	p3	Destry, Ralph
75	d8	Harris, Richard
	s8	Alazar, Pat
		Schaller, Paula
89	p3	Gilbert, Diane
	p8	Embry, Albert
	s8	Howser, Peter

Figure 7-1 Alphabetical Employee List by Job Code and Department

It takes three sorts to order the file. Start with the *least* important key (name), and lead up to the *most* important key, (department number). During sorting, dBASE II moves only as many records as it must. The following series of commands sorts the file into the order shown in Figure 7-1.

```
use employee
sort on name to namesort
use namesort
sort on job:code to codesort
use codesort
sort on dept:num to deptsort
use deptsort
list
```

```
. use employee
. sort on last:name to namesort
SORT COMPLETE
. use namesort
. sort on job:code to codesort
SORT COMPLETE
. use codesort
. sort on dept:num to deptsort
SORT COMPLETE
. use deptsort
. list
00001  Per      Inders   321 Sawtelle  Tuscon      AZ 85702          0.00 .F.
00002  Peter    Shaffer  43 Clinton Av Montclair   NJ 07042    d8 17900.00 .F.
00003  Mary     Berger   10 Kearney Rd Needham     MA 02194    p8     0.00 .F.
00004  Frank    Green    441 Spicer Av Tampa       FL 33622    s8 12500.00 .F.
00005  Paul     Rowland  709 Key St    Houston     TX 77007 16 p3 15750.00 .F.
00006  Gail     Smyth    817 Sth St.   Ambler      PA 19147 16 p4 20100.00 .F.
00007  Jean     Freitag  854 Moose Blv Houston     TX 77006 16 p9  2775.00 .F.
00008  Ralph    Destry   234 Mahogony  Deerfield   FL 33441 38 p3 15575.00 .F.
00009  Terry    Green    567 Doheny Dr Hollywood   CA 90044 54 m1 14500.00 .F.
00010  John     Brown    456 Minnow Pl Burlington  MA 01730 54 p3 21000.00 .F.
00011  Alice    Peters   676 Wacker Dr Chicago     IL 60606 54 s8 13700.00 .F.
00012  Richard  Harris   101 Enders Dr Syracuse    NY 13211 75 d8 21700.00 .F.
00013  Pat      Alazar   123 Crater    Everett     WA 98206 75 s8 12500.00 .F.
00014  Paula    Schaller 721 Spring St Everett     WA 98206 75 s8 11000.00 .F.
00015  Diane    Gilbert  280 Cactus Wy Las Cruces  NM 88001 89 p3 24500.00 .F.
00016  Albert   Embry    345 Sage Ave  Palo Alto   CA 94303 89 p8 22200.00 .F.
00017  Peter    Howser   678 Dusty Rd  Chicago     IL 60631 89 s8  9500.00 .F.
```

MODIFYING THE RECORD CONTENTS

The preceding commands add, delete, and sort whole records. dBASE II also provides several ways to change individual field values in single records and groups of records. EDIT, BROWSE, and CHANGE all allow you to modify individual records interactively. REPLACE changes selected fields in the entire data base with a single statement. UPDATE modifies one file with the contents of another.

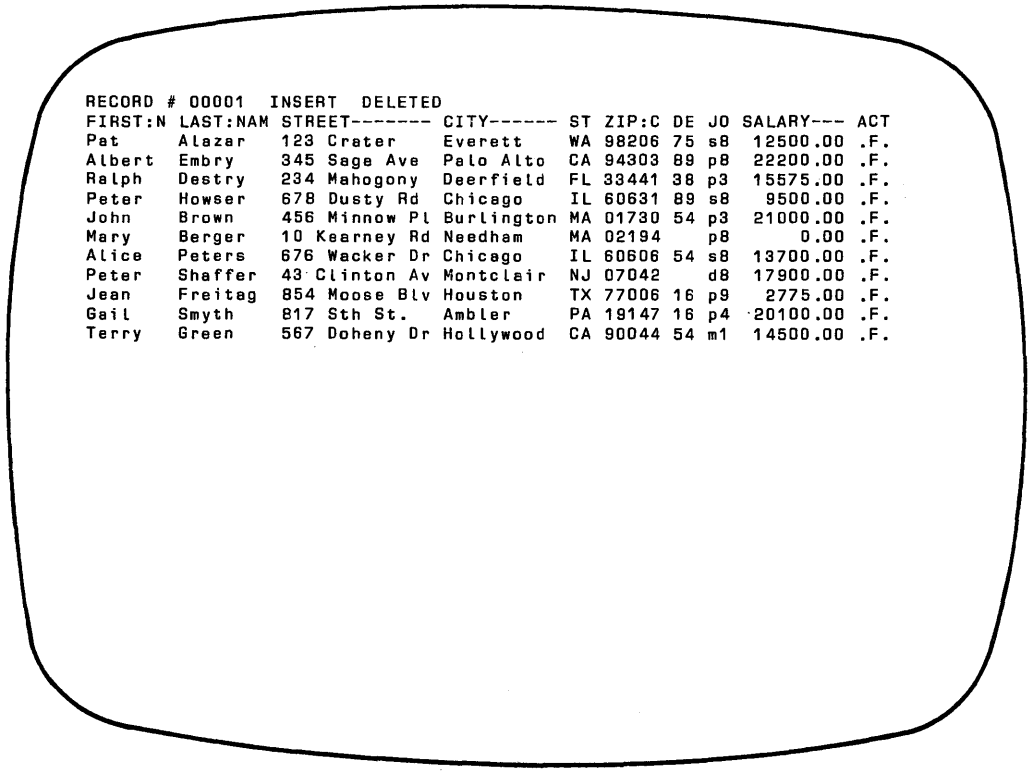
Interactive Modifications to Records

EDIT AND BROWSE

EDIT, explained in Chapter 5, allows you to edit one record at a time moving sequentially through the file. Like EDIT, BROWSE provides full-screen editing capabilities. However, BROWSE displays more than one record at a time. To see how this command works, enter:

```
use employee
browse
```

Your display will look similar to the one that follows.



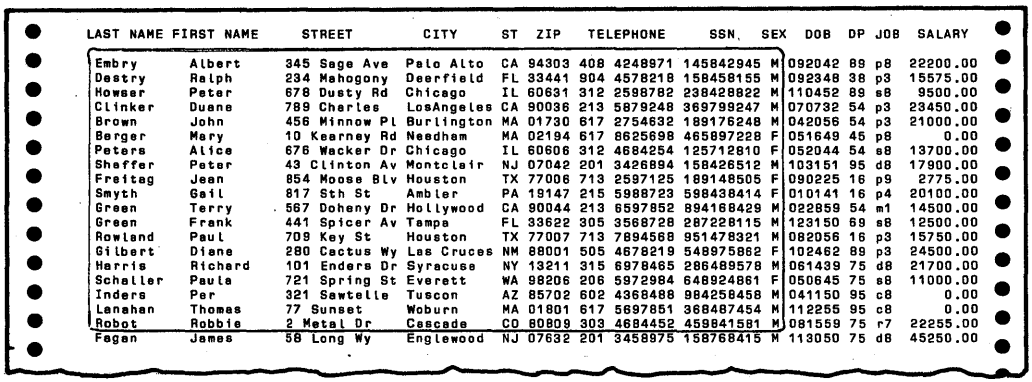
```
RECORD # 00001  INSERT  DELETED
FIRST:N LAST:NAM STREET------- CITY------ ST ZIP:C DE JO SALARY--- ACT
Pat     Alazar   123 Crater    Everett    WA 98206 75 s8  12500.00 .F.
Albert  Embry    345 Sage Ave  Palo Alto  CA 94303 89 p8  22200.00 .F.
Ralph   Destry   234 Mahogony  Deerfield  FL 33441 38 p3  15575.00 .F.
Peter   Howser   678 Dusty Rd  Chicago    IL 60631 89 s8   9500.00 .F.
John    Brown    456 Minnow Pl Burlington MA 01730 54 p3  21000.00 .F.
Mary    Berger   10 Kearney Rd Needham    MA 02194    p8      0.00 .F.
Alice   Peters   676 Wacker Dr Chicago    IL 60606 54 s8  13700.00 .F.
Peter   Shaffer  43 Clinton Av Montclair  NJ 07042    d8  17900.00 .F.
Jean    Freitag  854 Moose Blv Houston    TX 77006 16 p9   2775.00 .F.
Gail    Smyth    817 Sth St.   Ambler     PA 19147 16 p4  20100.00 .F.
Terry   Green    567 Doheny Dr Hollywood  CA 90044 54 m1  14500.00 .F.
```

The top line of the display is used for system messages. The current "RECORD#" appears first. If the system is in insert mode (**INS** or **CTRL V** has been pressed), the word "INSERT" appears next. If the record has been marked for deletion, "DELETED" appears as the last word on the message line.

The remainder of the display is a two-dimensional table that displays the contents of the data base file in pieces. Field names appear on the first row, either truncated or filled with dashes on the right as necessary so that the number of characters in the field name equals the field width. Data field values are displayed below the field name, one record per row.

BROWSE displays up to 19 records at a time, starting with the current record. The system displays as many data fields as possible at one time. If the record exceeds the

number of characters that can fit on one line of the display screen, you can see the display in pieces. For example, assume that Figure 7-2 represents a data base file. The circled area fills the display screen.



LAST NAME	FIRST NAME	STREET	CITY	ST	ZIP	TELEPHONE	SSN	SEX	DOB	DP	JOB	SALARY
Embry	Albert	345 Sage Ave	Palo Alto	CA	94303	408 4248971	145842945	M	092042	89	p8	22200.00
Destry	Ralph	234 Mahogony	Deerfield	FL	33441	904 4578218	158458155	M	092348	38	p3	15575.00
Howser	Peter	678 Dusty Rd	Chicago	IL	60631	312 2598782	238428822	M	110452	89	s8	9500.00
Clinker	Duane	789 Charles	LosAngeles	CA	90036	213 5879248	369799247	M	070732	54	p3	23450.00
Brown	John	456 Minnow Pl	Burlington	MA	01730	617 2754632	189176248	M	042056	54	p3	21000.00
Berger	Mary	10 Kearney Rd	Needham	MA	02194	617 8625698	465897228	F	051649	45	p8	0.00
Peters	Alice	676 Wacker Dr	Chicago	IL	60606	312 4684254	125712810	F	052044	54	s8	13700.00
Sheffer	Peter	43 Clinton Av	Montclair	NJ	07042	201 3426894	158426512	M	103151	95	d8	17900.00
Freitag	Jean	854 Moose Blv	Houston	TX	77006	713 2597125	189148505	F	090225	16	p9	2775.00
Smyth	Gail	817 Sth St	Ambler	PA	19147	215 5988723	598438414	F	010141	16	p4	20100.00
Green	Terry	567 Doheny Dr	Hollywood	CA	90044	213 6597852	894168429	M	022859	54	m1	14500.00
Green	Frank	441 Spicer Av	Tampa	FL	33622	305 3568728	287228115	M	123150	69	s8	12500.00
Rowland	Paul	709 Key St	Houston	TX	77007	713 7894568	951478321	M	082056	16	p3	15750.00
Gilbert	Diane	280 Cactus Wy	Las Cruces	NM	88001	505 4678219	548975862	F	102462	89	p3	24500.00
Harris	Richard	101 Enders Dr	Syracuse	NY	13211	315 6978465	286489578	M	061439	75	d8	21700.00
Schaller	Paula	721 Spring St	Everett	WA	98206	206 5972984	648924861	F	050645	75	s8	11000.00
Inders	Per	321 Sawtelle	Tuscon	AZ	85702	602 4368488	984258458	M	041150	95	c8	0.00
Lanahan	Thomas	77 Sunset	Woburn	MA	01801	617 5697851	368487454	M	112255	95	c8	0.00
Robot	Robbie	2 Metal Dr	Cascade	CO	80809	303 4684452	459841581	M	081559	75	r7	22255.00
Fegen	James	58 Long Wy	Englewood	NJ	07632	201 3458975	158768415	M	113050	75	d8	45250.00

Figure 7-2 Sample Data Base File for BROWSE Command

Think of the display screen as a window into the data base. You can pan right and left using **CTRL B** and **CTRL Z** respectively to see the remaining fields and return to the original display. You can scroll up and down using **CTRL R** and **CTRL C** to see other records (rows in the table).

Data can be edited using the full-screen edit commands listed in Table 7-1.

Table 7-1 Full Screen Edit Commands - BROWSE

COMMAND	ACTION
TAB ↓ **CTRL F** **CTRL J**	Moves cursor down to the next field
SHIFT TAB ↑ **CTRL A** **CTRL K**	Moves cursor up to the previous field
→ **CTRL D**	Moves cursor ahead one character

Table 7-1 Full Screen Edit Commands - BROWSE (cont'd)

COMMAND	ACTION
← **CTRL S**	Moves cursor back one character within a field, and back one field from the first character of a field
DEL **CTRL G** **CTRL X**	Deletes character under the cursor
CTRL Z	Pans window one field to the right
CTRL B	Pans window one field to the left
CTRL C	Saves changes and advances to the next record
CTRL R	Saves changes and backs up to the previous record with the cursor positioned on the first character of the field it was in when CTRL R was pressed.
CTRL W	Saves changes to current record and returns to dBASE II command mode
CTRL Q	Aborts changes in current record and returns to dBASE II command mode
CTRL Y	Erases field
CTRL U	Toggles record deletion mark on/off
INS **CTRL V**	Toggles insert/overtype modes
PRINT **CTRL P**	Toggles printer on/off

CHANGE

CHANGE allows you to make a number of alterations to a data base selectively. You supply the list of fields to be changed and, optionally, the conditions for which

records are to be selected. The system selects the records that meet the conditions and presents the fields to change in the order given in the command. You then have the choice of entering new data, modifying the current data, or skipping to the next field. When the field list has been exhausted, the system proceeds to the next record as specified in the scope. The format of the CHANGE command is:

CHANGE [<scope>] FIELD <field list> [FOR <exp>]

- The default for scope is NEXT 1 (the current record).
- The FIELD phrase, including the keyword, is required.
- To skip a field and leave the data intact, press **RETURN**.

The CHANGE command operates slightly differently for character type fields than for logical and numeric fields.

For character fields:

- To delete a field in its entirety, replace all data with spaces.
- When entering data, be sure the field does not exceed the number of spaces allocated in the width parameter of the record structure. The system does not indicate if the length is exceeded, but it does truncate characters on the right as necessary to fit the allowable space.
- After a change is entered, the system prompts you with "CHANGE?" to allow you to alter the field again before saving it.
- To abort the command, press **ESC**.

For logical and numeric fields:

- To delete a field in its entirety, press **CTRL Y** in response to "CHANGE?".
- Do not press **ESC** to abort the command. It resets a numeric field to zero and produces an error with logical fields.
- When entering data into numeric fields, be sure the data is all numeric. The system does not indicate if character data has been entered, but it does replace invalid numeric fields with zero.

Replacing Fields Quickly

REPLACE can be used to change one or more fields in some or all records in a data base. This command is very powerful because it automatically replaces each field named with whatever data is specified in every record that meets the conditions.

The format of the REPLACE command is:

REPLACE [<scope>] <field> WITH <data> [,<field2> WITH <data>...]
[FOR <exp>]

- If <scope> is not supplied in the command, REPLACE acts only on the current record.
- The keyword WITH is required for each field replacement. There is no limit to the number of fields that can be REPLACEd by one command except the dBASE II command line limit of 254 characters.
- The <data> can be a constant, variable, or expression.

Character fields must be replaced with character data, numeric fields with numeric data, and logical fields with logical data. Remember to enclose character constants between quotes or square brackets. The following example demonstrates how to REPLACE a logical field with a constant in every record in a file.

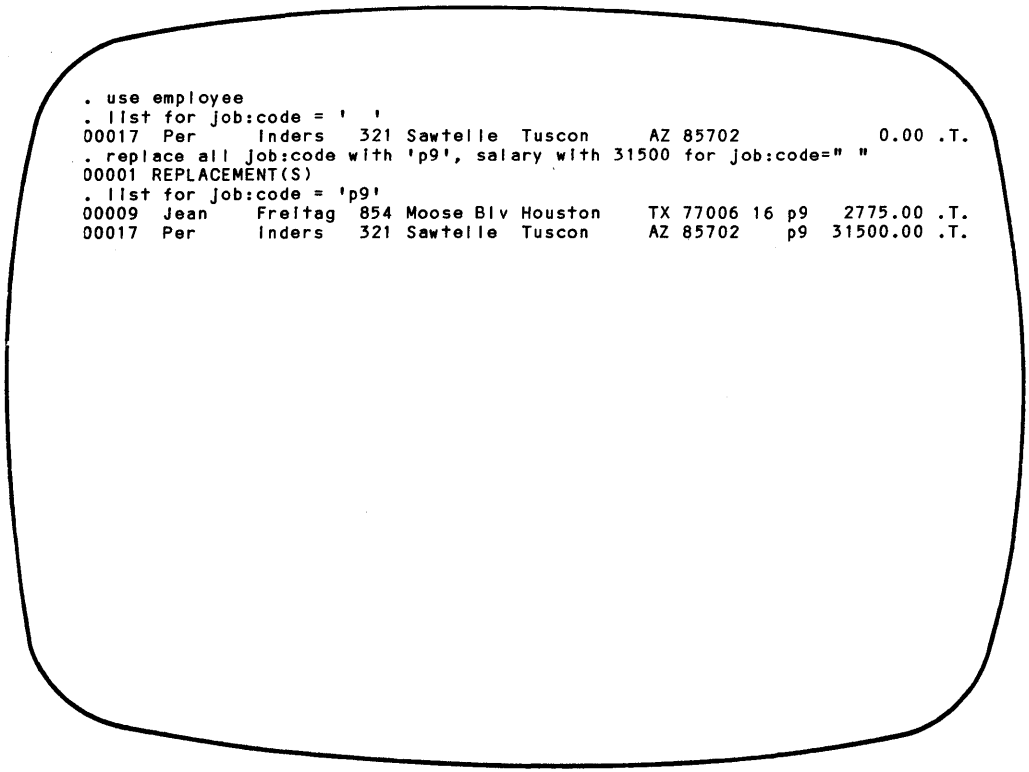
When the EMPLOYEE data base was created, the default value ".F." was supplied for the ACT:STAT field for every employee. It hasn't mattered up to this point since the field was never used. But when the field is used, it must have valid data. To change the value to ".Y." (for Active) for every employee, enter the following commands.

```
use employee
replace all act:stat with Y
list
```

```
. use employee
. replace all act:stat with y
00017 REPLACEMENT(S)
. list
00001  Pat      Alazar    123 Crater     Everett     WA 98206 75 s8  12500.00 .T.
00002  Albert   Embry     345 Sage Ave   Palo Alto   CA 94303 89 p8  22200.00 .T.
00003  Ralph    Destry    234 Mahogony   Deerfield   FL 33441 38 p3  15575.00 .T.
00004  Peter    Howser    678 Dusty Rd   Chicago     IL 60631 89 s8   9500.00 .T.
00005  John     Brown     456 Minnow Pl  Burlington  MA 01730 54 p3  21000.00 .T.
00006  Mary     Berger    10 Kearney Rd  Needham     MA 02194    p8      0.00 .T.
00007  Alice    Peters    676 Wacker Dr  Chicago     IL 60606 54 s8  13700.00 .T.
00008  Peter    Shaffer   43 Clinton Av  Montclair   NJ 07042    d8  17900.00 .T.
00009  Jean     Freitag   854 Moose Blv  Houston     TX 77006 16 p9   2775.00 .T.
00010  Gail     Smyth     817 Sth St.    Ambler      PA 19147 16 p4  20100.00 .T.
00011  Terry    Green     567 Doheny Dr  Hollywood   CA 90044 54 m1  14500.00 .T.
00012  Frank    Green     441 Spicer Av  Tampa       FL 33622    s8  12500.00 .T.
00013  Paul     Rowland   709 Key St     Houston     TX 77007 16 p3  15750.00 .T.
00014  Diane    Gilbert   280 Cactus Wy  Las Cruces  NM 88001 89 p3  24500.00 .T.
00015  Richard  Harris    101 Enders Dr  Syracuse    NY 13211 75 d8  21700.00 .T.
00016  Paula    Schaller  721 Spring St  Everett     WA 98206 75 s8  11000.00 .T.
00017  Per      Inders    321 Sawtelle   Tuscon      AZ 85702            0.00 .T.
```

You can also replace a field selectively with a constant, variable, or expression. To fill in the department number for those employee records in which it is missing, enter:

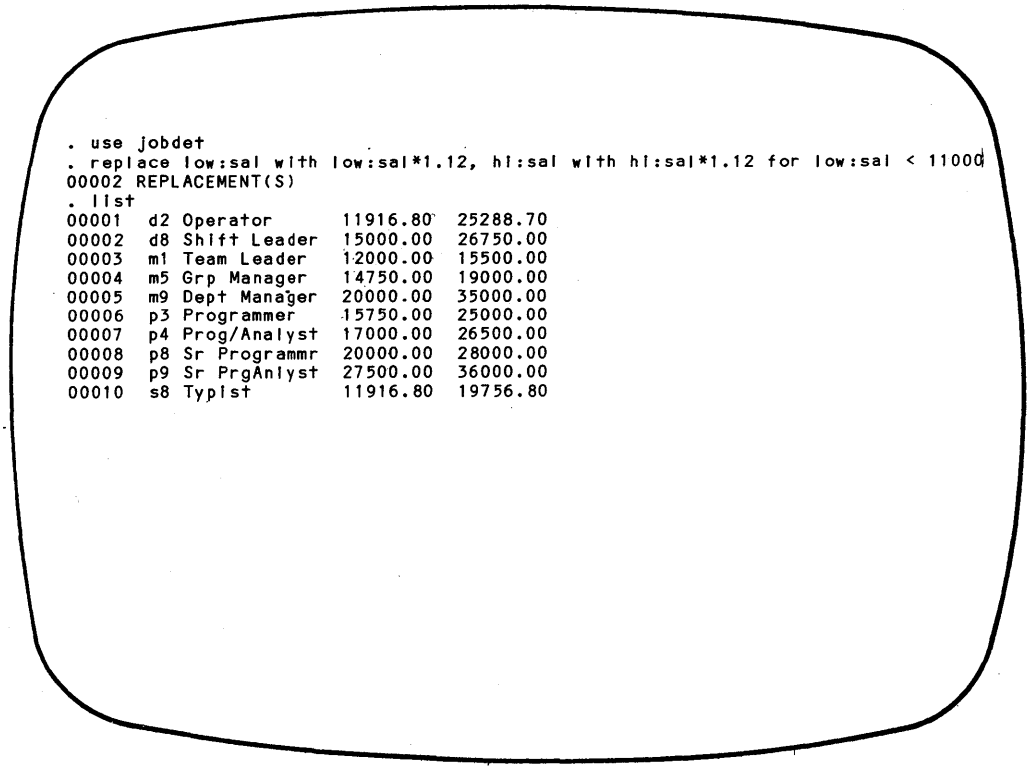
```
use employee
list for job:code = " "

replace all job:code with 'p9', salary with 31500 for job:code = " "
list for job:code "p9"
```



```
. use employee
. list for job:code = '  '
00017  Per      Inders    321 Sawtelle  Tuscon      AZ 85702              0.00 .T.
. replace all job:code with 'p9', salary with 31500 for job:code=" "
00001 REPLACEMENT(S)
. list for job:code = 'p9'
00009  Jean     Freitag   854 Moose Blv Houston     TX 77006 16 p9    2775.00 .T.
00017  Per      Inders    321 Sawtelle  Tuscon      AZ 85702    p9   31500.00 .T.
```

When the REPLACE command contains an expression, the system evaluates the expression and places the result in the specified field. For example, to raise the salary range by 12% for every job with the low salary less than $11,000, you would use the following command sequence.

```
use jobdet
replace low:sal with low:sal*1.12, hi:sal with hi:sal*1.12 for low:sal
<11000
list
```



```
. use jobdet
. replace low:sal with low:sal*1.12, hi:sal with hi:sal*1.12 for low:sal < 11000
00002 REPLACEMENT(S)
. list
00001   d2 Operator       11916.80  25288.70
00002   d8 Shift Leader   15000.00  26750.00
00003   m1 Team Leader    12000.00  15500.00
00004   m5 Grp Manager    14750.00  19000.00
00005   m9 Dept Manager   20000.00  35000.00
00006   p3 Programmer     15750.00  25000.00
00007   p4 Prog/Analyst   17000.00  26500.00
00008   p8 Sr Programmr   20000.00  28000.00
00009   p9 Sr PrgAnlyst   27500.00  36000.00
00010   s8 Typist         11916.80  19756.80
```

REPLACE is useful for completing records in which the value of one or more fields is calculated from other fields in the record. Table 7-2 defines the structure of a sample Orders File.

Table 7-2 Orders File

ORDERS				
CUST #	ITEM	QTY	UNIT PRICE	AMOUNT

When creating order data records, you can enter customer number, item, quantity, and unit price for each order received. To compute the AMOUNT, just use one command: REPLACE All AMOUNT WITH QTY * UNIT:PRICE. This command can be a real time saver.

Merging Records from two Data Bases

Data can be transferred from one data base file to another with the UPDATE command. The two files must have the same key field, and they must both be in key sequence before this command is issued. The system matches records from the two files based on key values. When a match is found, the file in USE is updated with field values from the corresponding record in the FROM file.

```
UPDATE FROM <file> ON <key> [ADD <field list>]
                            [REPLACE <field list>]
```

- The file in USE can be either indexed or sorted on the key.
- The FROM file must be sorted on the key.
- If ADD is specified, the data in the FROM fields are numerically *added* to the corresponding fields in the USE file.
- If REPLACE is specified, the FROM file values *replace* the corresponding fields in the USE record.

The following series of screens demonstrate how the UPDATE command works. An inventory file (INVENTRY) is updated with transactions in a file called INVUPDAT.

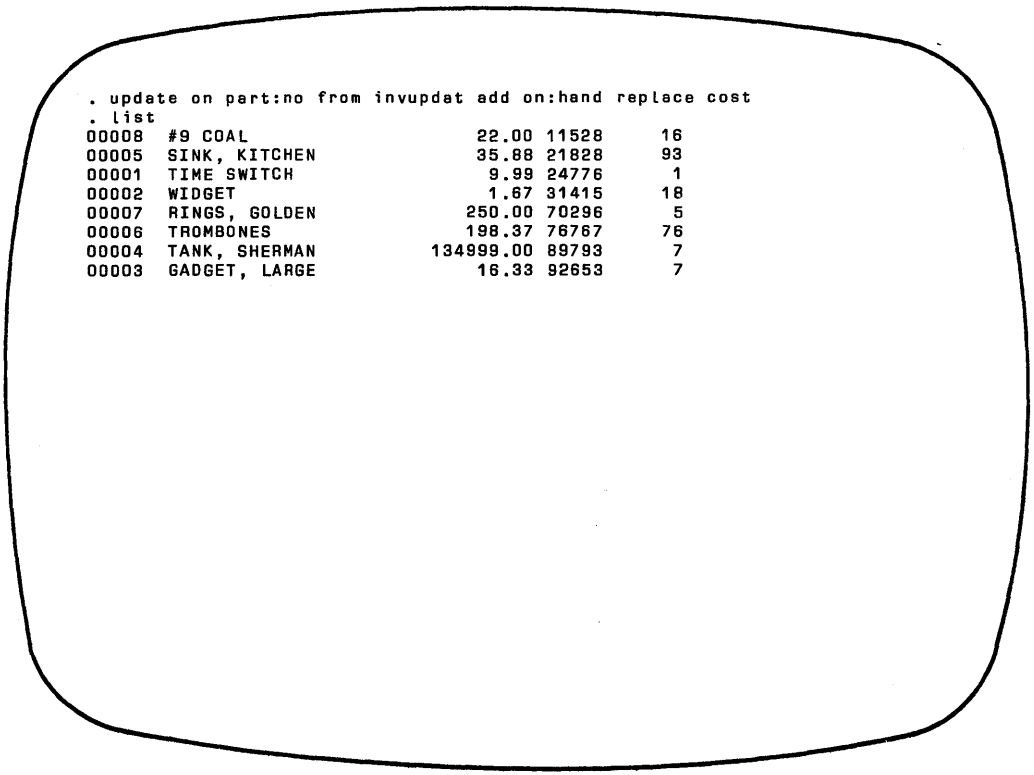
1. The file of update records is sorted on PART:NO.

```
. use invupdat
. display structure
STRUCTURE FOR FILE:  INVUPDAT.DBF
NUMBER OF RECORDS:   00003
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH    DEC
001     PART:NO       C    005
002     ON:HAND       N    005
003     COST          N    010     002
** TOTAL **             00021
. list
00001   21828     77        35.88
00002   70296      0       250.00
00003   89793      2    134999.00
```

2. The inventory file is indexed on PART:NO. The index file is called INVENTRY.NDX.

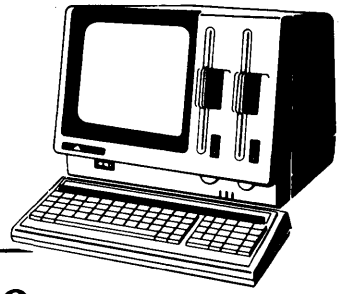
```
. use inventry index inventry
. display structure
STRUCTURE FOR FILE:  INVENTRY.DBF
NUMBER OF RECORDS:   00008
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH   DEC
001      ITEM          C    020
002      COST          N    010    002
003      PART:NO       C    005
004      ON:HAND       N    005
** TOTAL **             00041
. list    *
00008  #9 COAL                    22.00 11528     16
00005  SINK, KITCHEN              34.72 21828     16
00001  TIME SWITCH                 9.99 24776      1
00002  WIDGET                      1.67 31415     18
00007  RINGS, GOLDEN             200.00 70296      5
00006  TROMBONES                 198.37 76767     76
00004  TANK, SHERMAN          134999.00 89793      5
00003  GADGET, LARGE              16.33 92653      7
```

3. The inventory file, the file in USE, is updated with the contents of the transaction file.



```
. update on part:no from invupdat add on:hand replace cost
. list
00008  #9 COAL                    22.00 11528     16
00005  SINK, KITCHEN              35.88 21828     93
00001  TIME SWITCH                 9.99 24776      1
00002  WIDGET                      1.67 31415     18
00007  RINGS, GOLDEN             250.00 70296      5
00006  TROMBONES                 198.37 76767     76
00004  TANK, SHERMAN          134999.00 89793      7
00003  GADGET, LARGE              16.33 92653      7
```

The prices of the Golden Rings and Sherman Tanks were replaced. The quantities of Kitchen Sink and Sherman Tank were incremented by the values in the update file.



Chapter 8

Working With The Data Base Structure

As your information needs change, you are likely to find that your file structures need modification. New fields must be added. Unused fields, that only take up space and clutter the displays, can be deleted. Field characteristics — name, size, and type — may change. With dBASE II, it takes just a few commands to duplicate and modify file structures.

DUPLICATING FILES

It is a good idea to keep backup copies of important data files. To duplicate dBASE II files without exiting to the CP/M operating system, use the COPY command. To see how this works, enter:

```
use jobdet
copy to jobtemp
use jobtemp
display structure
list
```

As you can see on the screen that follows, this command sequence copies the entire JOBDET.DBF file, the record structure and all data records except those marked for deletion. The DISPLAY and LIST commands are not necessary for performing the COPY. They are included for demonstration purposes only.

```
. use jobdet
. copy to jobtemp
00010 RECORDS COPIED
. use jobtemp
. display structure
STRUCTURE FOR FILE:  JOBTEMP.DBF
NUMBER OF RECORDS:   00010
DATE OF LAST UPDATE: 11/04/82
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH   DEC
001     JOB:CODE       C    002
002     JOB:TITLE      C    012
003     LOW:SAL        N    008    002
004     HI:SAL         N    008    002
** TOTAL **             00031
. list
00001  d2 Operator      10640.00  22579.20
00002  d8 Shift Leader  15000.00  26750.00
00003  m1 Team Leader   12000.00  15500.00
00004  m5 Grp Manager   14750.00  19000.00
00005  m9 Dept Manager  20000.00  35000.00
00006  p3 Programmer    15750.00  25000.00
00007  p4 Prog/Analyst  17000.00  26500.00
00008  p8 Sr Programmr  20000.00  28000.00
00009  p9 Sr PrgAnlyst  27500.00  36000.00
00010  s8 Typist        10640.00  17640.00
```

Selective Copies

The general form of the COPY command is:

COPY TO <file> [<scope>] [FIELD <field list>] [FOR <exp>]

Since the command takes qualifying phrases, it can be used to copy whole files or to copy selectively. The command is written according to the following rules.

- The file name can be either a new or existing file.
- If the file does not exist, it is created.
- If the file does exist, its structure and contents are completely replaced by the structure and contents of the file in USE.

- The new file has the same file type extension (for example, ".DBF") as the file from which it is copied.
- The scope parameter defaults to ALL.
- If a list of fields is supplied, those data fields are the only ones copied to the new file. The keyword FIELD must be used. The field names in the field list must be separated by commas.
- Records marked for deletion are not copied.

DELETING AND RENAMING FILES

At this point, there are more files on your dBASE II diskette than you need. There are "original" files, files sorted into various sequences, backup copies, and so on.

It is a good practice to delete unnecessary files to free up space and avoid confusion. At the same time, it is a good idea to make backup copies of data files. To keep appropriately labelled copies of data files, use the COPY command with the DELETE and RENAME commands described in this section.

To remove files from the diskette, use the DELETE command.

DELETE FILE [<disk drive>:]<file>

Deleting files is a one-step process. Once this command is executed, the entire file is gone forever. It cannot be recalled. dBASE II includes two features that help prevent the accidental deletion of files. The keyword FILE must be entered, and you cannot DELETE a file that is currently in USE.

ABCORDER.DBF and TEMP.DBF are demonstration files created in Chapter 3 to hold PEOPLE records sorted in different orders. JOBDET.DBF is the job details file with records marked for deletion. Since you don't need these files and a few others, delete them by entering the following sequence of commands.

```
use
list files
delete file abcorder
delete file temp
delete file namesort
delete file codesort
delete file deptsort
delete file jobdet
list files
```

```
. use
. list files

DATABASE FILES   # RCDS    LAST UPDATE
PEOPLE   DBF     00006     00/00/00
TEMP     DBF     00006     10/21/82
ABCORDER DBF     00006     10/21/82
EMPLOYEE DBF     00017     10/31/82
NAMESORT DBF     00017     10/31/82
CODESORT DBF     00017     10/31/82
DEPTSORT DBF     00017     10/31/82
JOBDET   DBF     00010     00/00/00
JOBTEMP  DBF     00010     00/00/00


. delete file abcorder
FILE HAS BEEN DELETED
. delete file temp
FILE HAS BEEN DELETED
. delete file namesort
FILE HAS BEEN DELETED
. delete file codesort
FILE HAS BEEN DELETED
. delete file deptsort
FILE HAS BEEN DELETED
. delete file jobdet
FILE HAS BEEN DELETED
. list files

DATABASE FILES   # RCDS    LAST UPDATE
PEOPLE   DBF     00006     00/00/00
EMPLOYEE DBF     00017     10/31/82
JOBTEMP  DBF     00010     00/00/00
```

The USE command closed any open files. LIST FILES was included in the sequence to verify that the files were DELETEd. All the DELETEd files were data base files (type DBF) so it was not necessary to include the file type extension as part of the file name. For any other file types, the command requires the full file name. When files are not on the default drive, the drive specifier (disk drive:) must prefix the file name.

DELETE removes the file name from the CP/M directory and frees space on the diskette. Use this command to remove temporary files, SORT files, and any other unneeded files from diskettes.

Renaming Files

The file named JOBTEMP.DBF is the cleaned up version of the job details file. JOBDET.DBF is a better name for the file. It is more descriptive and is the name the file has been called throughout this guide. Since the JOBDET.DBF file has been deleted, you can change the name of JOBTEMP.DBF to JOBDET.DBF with the RENAME command.

RENAME <orig file> TO <new file>

This command changes the name of a file in the CP/M directory. If no file type extension is given, dBASE II assumes that the type is DBF. Although it is possible to change the file type along with the name, it is generally not a good practice to do so. Backups are the exception. The extension ".BAK" is conventionally used to identify backup copies of files.

Rename the JOBTEMP file to JOBDET as shown in the following screen.

```
. use jobtemp
. rename jobtemp to jobdet
. list files

DATABASE FILES   # RCDS     LAST UPDATE
PEOPLE   DBF     00006      00/00/00
EMPLOYEE DBF     00017      10/31/82
JOBDET   DBF     00010      00/00/00
```

WARNING

Do not RENAME a file that is in USE.

Maximizing Diskette Space

The PACK command described in Chapter 7 removes deleted records from a file but does not release the freed space on the diskette. To release the space, use the following procedure:

1. USE the original file. This is the file that has been PACKed or that has records marked for deletion.
2. COPY the original file to a new file.
3. DELETE the original file.
4. RENAME the new file with the original file name.

Backup Procedures for a Dual-Drive APC

With the three commands just described, you can create backup copies of individual files with appropriate names. If you have a dual-drive APC, you can create backup data files on a diskette kept for just that purpose. Individual files can be copied from within dBASE II. The following is a suggested backup sequence for a dual-drive system.

1. If the data file to be copied is on Drive A with the dBASE II program, skip this step. Otherwise, COPY B:<file> TO A:<temp>.
2. Insert the formatted backup diskette in Drive B.
3. COPY A:<temp> TO B:<backup>
4. Remove the backup diskette from Drive B.
5. Take appropriate action with the original file. This can mean doing nothing, renaming the file, or deleting it.
6. If the data file was copied to Drive A in Step 1, delete it: DELETE A:<temp>.

As an alternative, you can create backup copies of files using the CP/M-86 Peripherals Interchange Program (PIP). To use this procedure, you must exit from dBASE II. (See the CP/M-86 System User's Guide for information on how to use this program.)

Backup Procedures for a Single-Drive APC

If you have a single-drive APC, you must exit from dBASE II and use the CP/M-86 Peripherals Interchange Program (PIP1) to copy files from one diskette to another. (See the *CP/M-86 Systems User's Guide* for information on how to use this program.)

CHANGING THE DATA BASE STRUCTURE

The structure of a data base and its contents are two separate, but related, entities. So far, you have used commands to manipulate either the data base contents -records - or the entire data base file. This chapter describes procedures for working with the data base structure.

There are many ways to add and delete fields in a data base structure. The best method is the one that has the lowest probability of error and results in the least work for you. All methods should involve the same basic steps. Start by creating a new data base *structure* in a temporary file. Then, bring the *data* from the original file into the new, modified structure.

Generating the Structure

You can define the new structure with the CREATE command or COPY the structure of an existing file and MODIFY it.

CREATING THE STRUCTURE FROM SCRATCH

Assume you have the job of compiling a telephone list of all employees in the personnel data base. To do this, you could CREATE a new file, PHONELST, consisting of four fields: last name, first name, department number, and telephone number.

USING EXISTING DATA

Then, instead of entering name and department number data from the keyboard, you could copy the values from the EMPLOYEE file using a new form of APPEND. (You would have to fill in the telephone numbers another way.)

APPEND FROM <file> [FOR <exp>]

The records to be APPENDed are taken from an existing file.

dBASE II reads the structures of both files and looks for fields with matching names. It then transfers data from the "FROM" file to the file that is in USE for the fields that match.

This sequence is illustrated in the following screen.

```
. create phonelst
ENTER RECORD STRUCTURE AS FOLLOWS:
 FIELD    NAME,TYPE,WIDTH,DECIMAL PLACES
 001      last:name,c,8
 002      first:name,c,7
 003      dept:num,c,2
 004      phonenum,c,12
 005
INPUT DATA NOW? N
. use phonelst
. append from employee
00017 RECORDS ADDED
. list
00001  Alazar   Pat      75
00002  Embry    Albert   89
00003  Destry   Ralph    38
00004  Howser   Peter    89
00005  Brown    John     54
00006  Berger   Mary
00007  Peters   Alice    54
00008  Shaffer  Peter
00009  Freitag  Jean     16
00010  Smyth    Gail     16
00011  Green    Terry    54
00012  Green    Frank
00013  Rowland  Paul     16
00014  Gilbert  Diane    89
00015  Harris   Richard  75
00016  Schaller Paula    75
00017  Inders   Per
```

APPEND FROM works selectively when the optional [FOR <exp>] phrase is included. To APPEND the names of only those employees in department 54, you could use the command:

append from employee for dept:num='54'

When the FOR phrase is used, dBASE II APPENDs the records in the FROM file one by one, each time checking to see if the condition in the FOR expression is true. That is, the first record is appended. Then the expression is evaluated. If the expression is true, the record is kept. If the expression is false, the record is discarded. This procedure continues until the end-of-file is reached for the FROM file. This means that the fields used in the expression must reside in the file receiving the new records.

MODIFYING AN EXISTING STRUCTURE

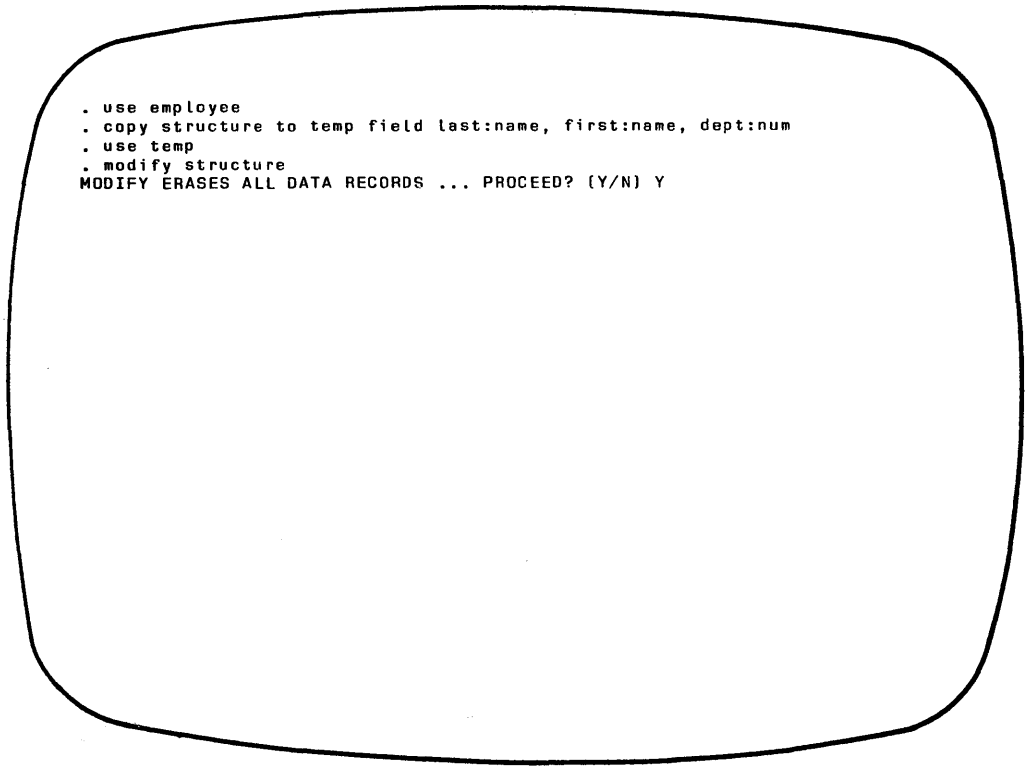
When only one or two fields from an existing data base are needed in the new file, it is efficient to CREATE the new structure. However, the sequence of COPY STRUCTURE and then MODIFY STRUCTURE is probably the fastest and easiest way to add, delete, rename, resize, or otherwise change a data base file structure.

The following format of the COPY command duplicates a file structure.

COPY STRUCTURE TO <file> [FIELD <field list>]

All or part of the structure can be copied. To copy selectively, use the optional FIELD phrase. List the names of fields from the original file that are to be included in the new structure. Commas are required in the field list to separate names.

As an alternative to the CREATE and APPEND sequence described previously, you can use COPY STRUCTURE to copy only the last name, first name, and department number fields from the EMPLOYEE file structure, as shown in the following screen.



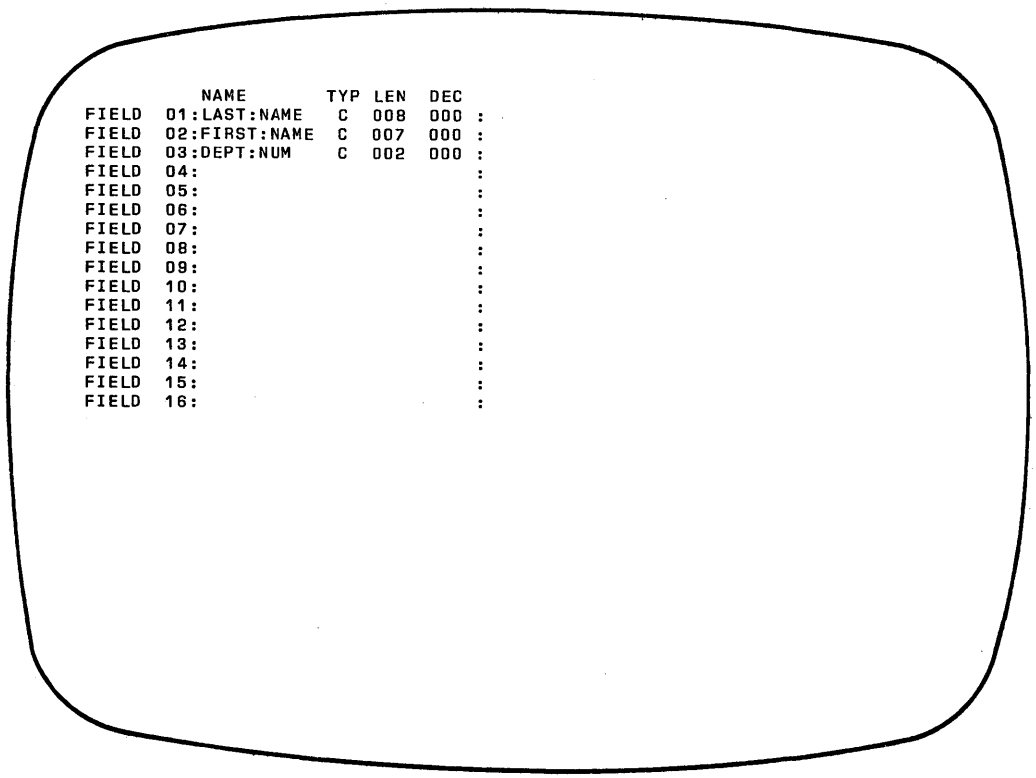
```
. use employee
. copy structure to temp field last:name, first:name, dept:num
. use temp
. modify structure
MODIFY ERASES ALL DATA RECORDS ... PROCEED? (Y/N) Y
```

This copies only two fields from the structure of the EMPLOYEE file. No data records are copied. Once the structure has been copied without data records, it can be changed with the MODIFY STRUCTURE command.

WARNING

The MODIFY STRUCTURE command deletes all data records that were in the USE file prior to the MODIFY. It should only be used with empty data bases. Use this command only after copying the structure.

MODIFY STRUCTURE acts upon the data base currently in USE. Since it destroys all data records, the system prompts you to be sure this is what you want to do before it takes any action. If you respond that you want to continue, the display screen is erased and up to 16 fields of the existing structure are displayed.



```
          NAME        TYP  LEN   DEC
FIELD  01:LAST:NAME    C   008   000 :
FIELD  02:FIRST:NAME   C   007   000 :
FIELD  03:DEPT:NUM     C   002   000 :
FIELD  04:                           :
FIELD  05:                           :
FIELD  06:                           :
FIELD  07:                           :
FIELD  08:                           :
FIELD  09:                           :
FIELD  10:                           :
FIELD  11:                           :
FIELD  12:                           :
FIELD  13:                           :
FIELD  14:                           :
FIELD  15:                           :
FIELD  16:                           :
```

You can make changes in full-screen edit mode using the commands listed in Table 8-1. Two commands are especially useful:

CTRL N inserts a blank line under the cursor, allowing you to insert fields where you want them;

CTRL T deletes the line under the cursor.

In the full-screen edit mode, you can add and delete fields and change field sizes and types. Exit from MODIFY STRUCTURE in either of two ways:

CTRL W changes the structure on disk;

CTRL Q aborts the command without making the changes, leaving the existing structure description intact.

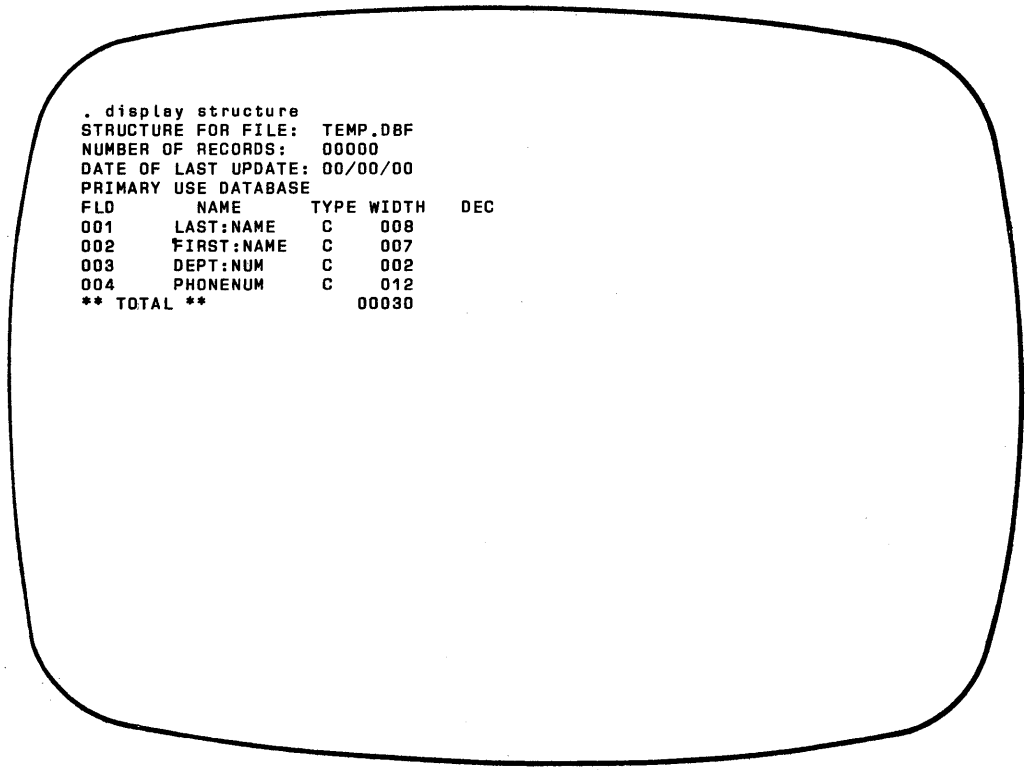
Table 8-1 Full Screen Edit Commands - MODIFY STRUCTURE

COMMAND	ACTION
TAB ↓ **CTRL F** **CTRL J**	Moves cursor down to the next field
SHIFT TAB ↑ **CTRL A** **CTRL K**	Moves cursor up to the previous field
→ **CTRL D**	Moves cursor ahead one character
← **CTRL S**	Moves cursor back one character within a field, and back one field from the first character of a field
DEL **CTRL G** **CTRL X**	Deletes character under the cursor
CTRL C	Advances to the next panel of fields
CTRL R	Moves back to the previous panel of fields
CTRL W	Saves changes to record structure and returns to dBASE II command mode

Table 8-1 Full Screen Edit Commands - MODIFY STRUCTURE (cont'd)

COMMAND	ACTION
CTRL Q	Returns to dBASE II command mode and does not change record structure
CTRL Y **CTRL N**	Erases field Inserts a blank line and moves subsequent fields down one line
CTRL T	Deletes line under the cursor and moves subsequent fields up one line
INS **CTRL V**	Toggles insert/overtype modes
PRINT **CTRL P**	Toggles printer on/off

The following screens demonstrate how MODIFY STRUCTURE is used to create the telephone list file. Once in full-screen edit mode, the first and last name fields are reversed by typing over the existing entries. Then, the new field-phone number-is added. When **CTRL W** is pressed, the new structure is saved and the system returns to command mode.



```
. display structure
STRUCTURE FOR FILE:  TEMP.DBF
NUMBER OF RECORDS:   00000
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH    DEC
001      LAST:NAME     C     008
002      FIRST:NAME    C     007
003      DEPT:NUM      C     002
004      PHONENUM      C     012
** TOTAL **             00030
```

To fill the last name, first name, and department number fields, data are appended from the EMPLOYEE file. As in the previous example, the telephone number must be supplied from another source.

```
. append from employee
00017 RECORDS ADDED
. list
00001  Alazar   Pat      75
00002  Embry    Albert   89
00003  Destry   Ralph    38
00004  Howser   Peter    89
00005  Brown    John     54
00006  Berger   Mary
00007  Peters   Alice    54
00008  Shaffer  Peter
00009  Freitag  Jean     16
00010  Smyth    Gail     16
00011  Green    Terry    54
00012  Green    Frank
00013  Rowland  Paul     16
00014  Gilbert  Diane    89
00015  Harris   Richard  75
00016  Schaller Paula    75
00017  Inders   Per
```

RESTRUCTURING THE SAMPLE DATA BASE

This section demonstrates how dBASE II commands can be used to restructure a data base. Follow the examples presented and make the same changes to your personnel data base.

The following changes are to be made to the personnel data base.

- Add a three digit employee number. This number will be used for seniority ranking and indicates hiring order.
- Increase the length of the last name field to 12 characters.

- Reverse the order of first name and last name.
- Sequence the file in alphabetical order for last name.
- Allow the following values for status: Active, Resigned/Retired, Terminated, Medical Leave. Change the field name to STATUS and the type from Logical to Character.

Using the commands discussed so far in this chapter, you can do everything required except change the name of the ACT:STAT field. The first step is to identify exactly what has to change. To do this, define what the data base should look like after all changes are in place. Then you can work backwards to figure out how to get there.

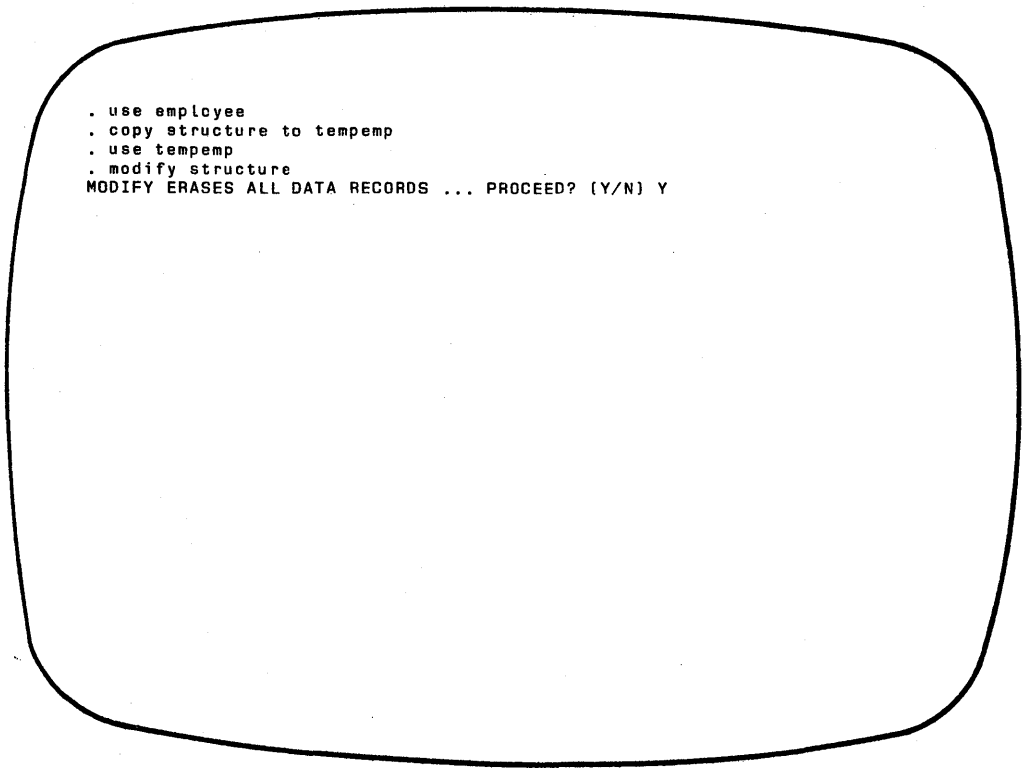
Table 8-2 describes the structure of EMPLOYEE after the changes. The JOBDET file does not change.

Table 8-2 New EMPLOYEE Record Structure

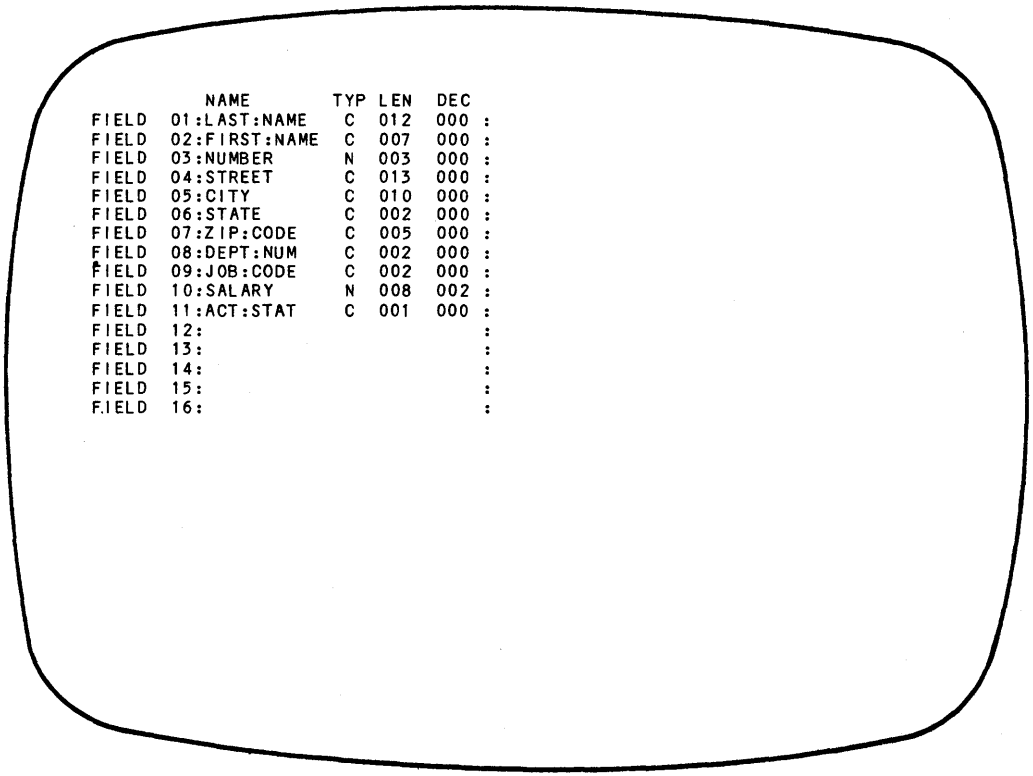
FIELD NAME	TYPE	WIDTH	DECIMAL PLACES
LAST:NAME	C	12	
FIRST:NAME	C	7	
NUMBER	N	3	
STREET	C	13	
CITY	C	10	
STATE	C	2	
ZIP:CODE	C	5	
DEPT:NUM	C	2	
JOB:CODE	C	2	
SALARY	N	8	2
STATUS	C	1	

There are many ways to make the changes. One sequence is shown in the following screens.

1. COPY the EMPLOYEE file structure to a temporary EMPLOYEE file called TEMPEMP. Use TEMPEMP and modify the structure.



2. In full-screen edit mode, insert the NUMBER field, reverse LAST:NAME and FIRST:NAME, change the length of LAST:NAME field, and change the type of ACT:STAT.



```
          NAME          TYP LEN  DEC
FIELD  01:LAST:NAME      C  012  000 :
FIELD  02:FIRST:NAME     C  007  000 :
FIELD  03:NUMBER         N  003  000 :
FIELD  04:STREET         C  013  000 :
FIELD  05:CITY           C  010  000 :
FIELD  06:STATE          C  002  000 :
FIELD  07:ZIP:CODE       C  005  000 :
FIELD  08:DEPT:NUM       C  002  000 :
FIELD  09:JOB:CODE       C  002  000 :
FIELD  10:SALARY         N  008  002 :
FIELD  11:ACT:STAT       C  001  000 :
FIELD  12:                           :
FIELD  13:                           :
FIELD  14:                           :
FIELD  15:                           :
FIELD  16:                           :
```

3. Press **CTRL W** to save the changes and return to command mode.

4. APPEND the data from the original EMPLOYEE file to TEMPEMP.
5. REPLACE the NUMBER field (currently the default value zero for all employees) with the RECORD #. REPLACE ACT:STAT with "A" (for Active) for all employees.
6. Verify the changes.

```
. append from employee
00017 RECORDS ADDED
. replace all number with #
00017 REPLACEMENT(S)
. replace all act:stat with "A"
00017 REPLACEMENT(S)
. list
00001  Alazar       Pat         1 123 Crater    Everett     WA 98206 75 s8
12500.00 A
00002  Embry        Albert      2 345 Sage Ave  Palo Alto   CA 94303 89 p8
22200.00 A
00003  Destry       Ralph       3 234 Mahogony  Deerfield   FL 33441 38 p3
15575.00 A
00004  Howser       Peter       4 678 Dusty Rd  Chicago     IL 60631 89 s8
9500.00 A
00005  Brown        John        5 456 Minnow Pl Burlington MA 01730 54 p3
21000.00 A
00006  Berger       Mary        6 10 Kearney Rd Needham     MA 02194 89 p8
30000.00 A
00007  Peters       Alice       7 676 Wacker Dr Chicago     IL 60606 54 s8
13700.00 A
00008  Shaffer      Peter       8 43 Clinton Av Montclair   NJ 07042 38 d8
17900.00 A
00009  Freitag      Jean        9 854 Moose Blv Houston     TX 77006 16 p9
2775.00 A
00010  Smyth        Gail       10 817 Sth St.   Ambler      PA 19147 16 p4
20100.00 A
00011  Green        Terry      11 567 Doheny Dr Hollywood   CA 90044 54 m1
14500.00 A
00012  Green        Frank      12 441 Spicer Av Tampa       FL 33622 54 s8
12500.00 A
00013  Rowland      Paul       13 709 Key St    Houston     TX 77007 16 p3
15750.00 A
00014  Gilbert      Diane      14 280 Cactus Wy Las Cruces  NM 88001 89 p3
```

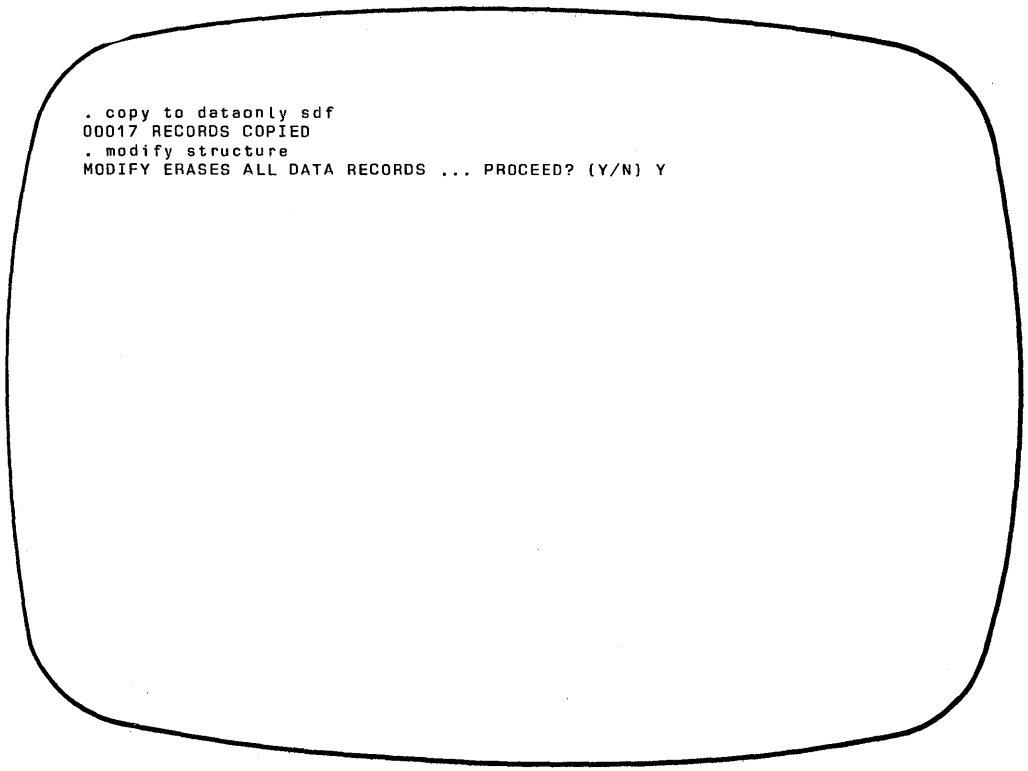
7. SORT the temporary file back to the EMPLOYEE file on LAST: NAME.
8. DELETE the temporary file.

CHANGING FIELD NAMES

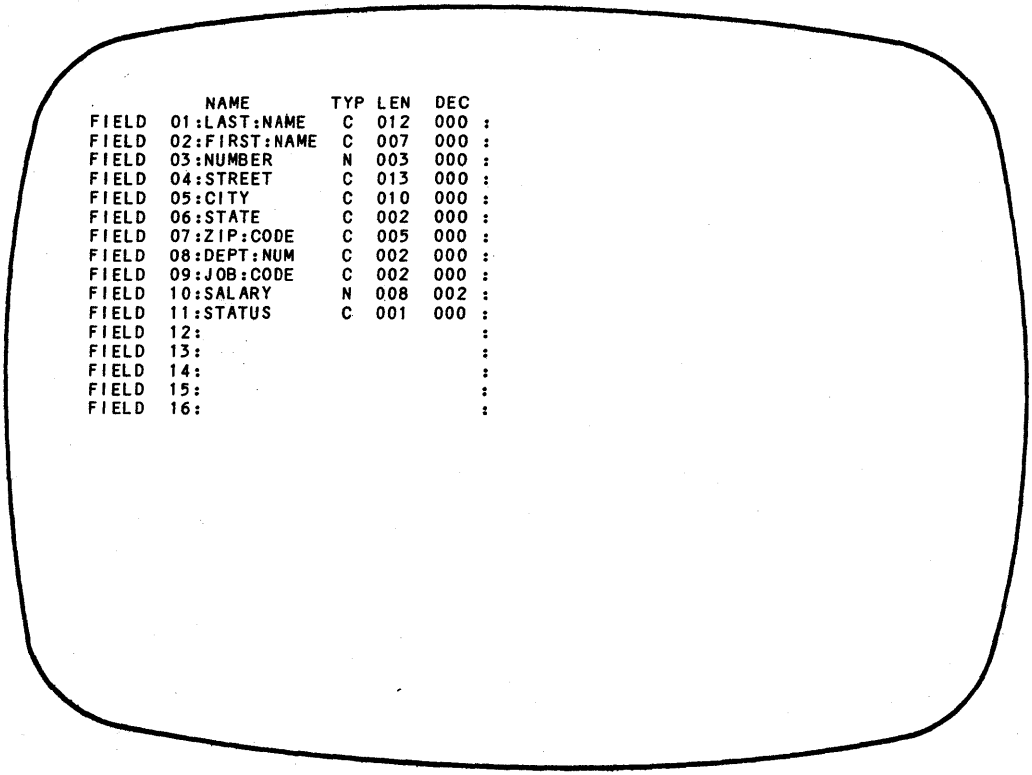
Renaming fields requires a different procedure than making any other changes to the data base structure. You can not just use the MODIFY STRUCTURE command to edit a field name because that destroys all existing records. And APPEND, as you know it, transfers data from one file to another only for matching field names. So that can't be used either. Instead, you have to separate the data records from the file structure, modify the structure, and then add the data back to the file without reference to field names.

Both COPY and APPEND have an option that allows you to do just that. To rename the field ACT:STAT to STATUS in EMPLOYEE.DBF, use the following procedure:

1. USE the file to be changed.
2. COPY the data records to a temporary file using the optional phrase SDF in the command.
3. Delete the original data records and enter full-screen edit mode by entering the MODIFY STRUCTURE command.



4. Now edit the field name only. Do not change the field width or type.



```
          NAME          TYP LEN  DEC
FIELD  01:LAST:NAME      C  012  000 :
FIELD  02:FIRST:NAME     C  007  000 :
FIELD  03:NUMBER         N  003  000 :
FIELD  04:STREET         C  013  000 :
FIELD  05:CITY           C  010  000 :
FIELD  06:STATE          C  002  000 :
FIELD  07:ZIP:CODE       C  005  000 :
FIELD  08:DEPT:NUM       C  002  000 :
FIELD  09:JOB:CODE       C  002  000 :
FIELD  10:SALARY         N  008  002 :
FIELD  11:STATUS         C  001  000 :
FIELD  12:                           :
FIELD  13:                           :
FIELD  14:                           :
FIELD  15:                           :
FIELD  16:                           :
```

5. Last, append the data records you copied back onto the original file with the new structure and delete the temporary file.

```
. append from dataonly sdf
00017 RECORDS ADDED
. delete file dataonly.txt
FILE HAS BEEN DELETED
. list structure
STRUCTURE FOR FILE:  EMPLOYEE.DBF
NUMBER OF RECORDS:   00017
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH   DEC
001      LAST:NAME     C    012
002      FIRST:NAME    C    007
003      NUMBER        N    003
004      STREET        C    013
005      CITY          C    010
006      STATE         C    002
007      ZIP:CODE      C    005
008      DEPT:NUM      C    002
009      JOB:CODE      C    002
010      SALARY        N    008    002
011      STATUS        C    001
** TOTAL **            00066
```

```
. list
00001  Alazar       Pat        1 123 Crater     Everett     WA 98206 75 s8
12500.00 A
00002  Berger       Mary       6 10 Kearney Rd  Needham     MA 02194 89 p8
30000.00 A
00003  Brown        John       5 456 Minnow Pl  Burlington  MA 01730 54 p3
21000.00 A
00004  Destry       Ralph      3 234 Mahogony   Deerfield   FL 33441 38 p3
15575.00 A
00005  Embry        Albert     2 345 Sage Ave   Palo Alto   CA 94303 89 p8
22200.00 A
00006  Freitag      Jean       9 854 Moose Blv  Houston     TX 77006 16 p9
2775.00 A
00007  Gilbert      Diane     14 280 Cactus Wy  Las Cruces  NM 88001 89 p3
24500.00 A
00008  Green        Terry     11 567 Doheny Dr  Hollywood   CA 90044 54 m1
14500.00 A
00009  Green        Frank     12 441 Spicer Av  Tampa       FL 33622 54 s8
12500.00 A
00010  Harris       Richard   15 101 Enders Dr  Syracuse    NY 13211 75 d8
21700.00 A
00011  Howser       Peter      4 678 Dusty Rd   Chicago     IL 60631 89 s8
9500.00 A
00012  Inders       Per       17 321 Sawtelle   Tuscon      AZ 85702 66 p9
31500.00 A
00013  Peters       Alice      7 676 Wacker Dr  Chicago     IL 60606 54 s8
13700.00 A
00014  Rowland      Paul      13 709 Key St     Houston     TX 77007 16 p3
15750.00 A
00015  Schaller     Paula     16 721 Spring St  Everett     WA 98206 75 s8
11000.00 A
00016  Shaffer      Peter      8 43 Clinton Av  Montclair   NJ 07042 38 d8
17900.00 A
00017  Smyth        Gail      10 817 Sth St.    Ambler      PA 19147 16 p4
20100.00 A
```

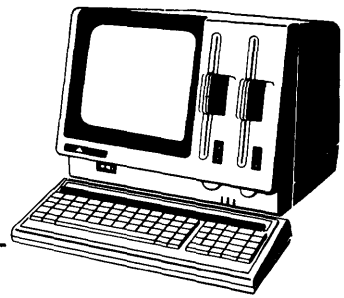
The above sequence uses the optional phrase [SDF] with the COPY and APPEND commands. *SDF* stands for Standard Data Format. CP/M files created in this format by dBASE II, word processors, or other programs can be read by dBASE II. Similarly, dBASE II, word processors, and other programs can create files in this format. This is how "foreign" files can be input to and output from dBASE II. (See Chapter 20 for more about CP/M files.)

COPY with the SDF option copies only the data records to the named file. (SDF and STRUCTURE are mutually exclusive options.) dBASE II automatically appends the ".TXT" extension to the file name. The data is stored by record with a carriage return and line feed signalling end-of-record. It is stored without any structure information, not even field names. TXT files can be edited with a word processor, but this can be dangerous if you plan to APPEND the TXT file to a DBF file structure.

When SDF is present in the APPEND command, the records are assumed to be in the Standard Data Format.

WARNING

Do not change field positions or sizes in the data base structure to which the TXT file will be APPENDed. The TXT data is saved by position, not by name. If you modify the structure in any way except changing field names, you will destroy your data base when you bring the saved data back into it.



Chapter 9

Indexed Files

Data base files contain data that can be organized and viewed from many perspectives. At different times, the EMPLOYEE file may need to be organized by zip code for mailings, by department number for staff reporting, and by employee number within status for seniority listings. dBASE II provides two ways to reorganize whole files.

SORT creates a new copy of the data base file for each sort sequence. The same information appears in every file; only the order differs. This takes up a lot of space on a diskette. Moreover, it is up to you to maintain the files. Additions must be inserted in the right place in each file or every file must be resorted after additions to insert the new records in the correct positions.

An alternative to SORT is INDEX. The file appears to be sorted, but actually only an index to the file is sorted. The INDEX command creates an index file consisting of one record for each record in the data base file. Index file records are sorted in ascending order on the value of the key field. Records consist of the key value and a pointer to the corresponding data record in the data base file. A data base file can have an unlimited number of index files associated with it.

This chapter describes indexing commands. It also describes how indexed files affect the operation of some commands that were discussed in previous chapters.

INDEXING FILES

Index files are sorted into ascending order according to the ASCII collating sequence (see Appendix F). Each index to a file must be created separately with the following command:

INDEX ON <exp> TO <index file>

The command creates or updates an index file with the extension ".NDX". All of the command fields are required.

- <exp> is known as the *key*. Its value determines the sequence of records in the index file. The expression can be a numeric variable, a character variable, or a simple or complex expression.
- The <index file> name must conform to the file name conventions described in Chapter 5. Do not include the extension. dBASE II automatically adds ".NDX" to the file name. Use the drive specifier prefix as necessary for files that are not on the default drive.

The key field can be a maximum of 100 characters. Keys can be formed by concatenating fields using the string operators (see the next section of this chapter). The TRIM function (described in Chapter 15) must not be used as part of an index key.

To index the EMPLOYEE file by department number, USE the file and issue the INDEX command. Then, to utilize the index feature, USE the data base file with the index.

```
use employee
index on dept:num to deptndx
use employee index deptndx
list
```

```
. use employee
. index on dept:num to deptndx
00017 RECORDS INDEXED
. use employee index deptndx
. list
00006  Freitag      Jean       9 854 Moose Blv Houston     TX 77006 16 p9
2775.00 A
00014  Rowland      Paul      13 709 Key St    Houston     TX 77007 16 p3
15750.00 A
00017  Smyth        Gail      10 817 5th St.   Ambler      PA 19147 16 p4
20100.00 A
00004  Destry       Ralph      3 234 Mahogony  Deerfield   FL 33441 38 p3
15575.00 A
00016  Shaffer      Peter      8 43 Clinton Av Montclair   NJ 07042 38 d8
17900.00 A
00003  Brown        John       5 456 Minnow Pl Burlington MA 01730 54 p3
21000.00 A
00008  Green        Terry     11 567 Doheny Dr Hollywood   CA 90044 54 m1
14500.00 A
00009  Green        Frank     12 441 Spicer Av Tampa       FL 33622 54 s8
12500.00 A
00013  Peters       Alice      7 676 Wacker Dr Chicago     IL 60606 54 s8
13700.00 A
00012  Inders       Per       17 321 Sawtelle  Tuscon      AZ 85702 66 p9
31500.00 A
00001  Alazar       Pat        1 123 Crater    Everett     WA 98206 75 s8
12500.00 A
00010  Harris       Richard   15 101 Enders Dr Syracuse    NY 13211 75 d8
21700.00 A
00015  Schaller     Paula     16 721 Spring St Everett     WA 98206 75 s8
11000.00 A
00002  Berger       Mary       6 10 Kearney Rd Needham     MA 02194 89 p8
30000.00 A
00005  Embry        Albert     2 345 Sage Ave  Palo Alto   CA 94303 89 p8
```

Notice that when the file is USEd with the index, the records appear in department number order but retain their original record sequence numbers.

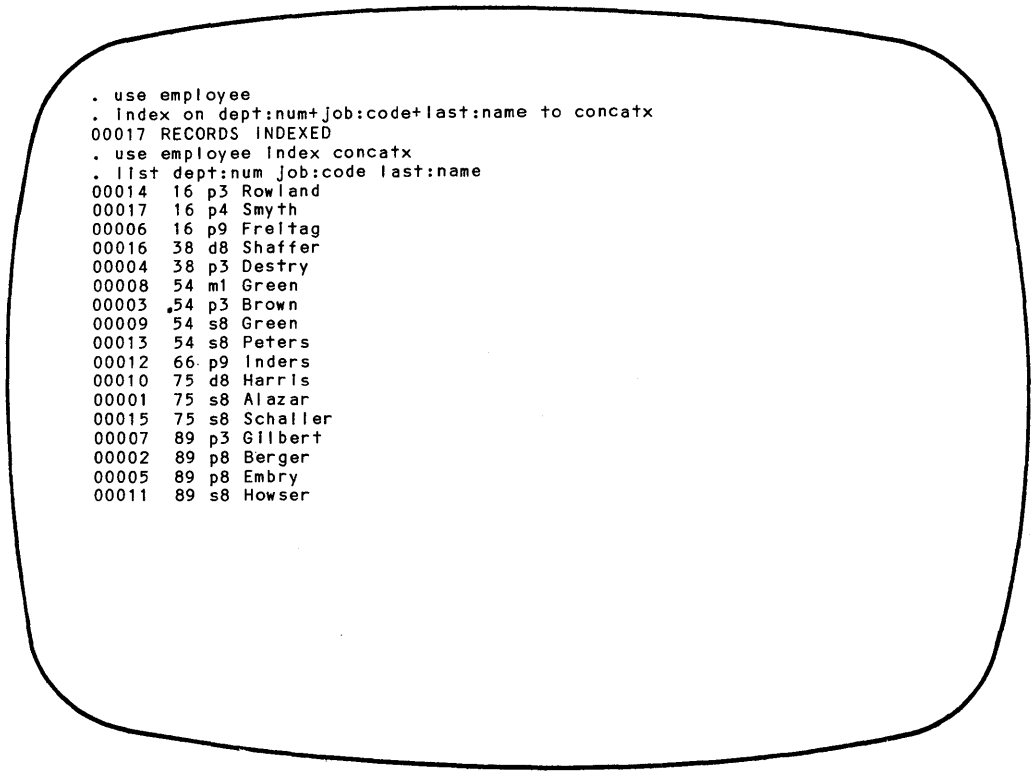
INDEXING ON MORE THAN ONE FIELD

To index on several keys at once, concatenate the key fields with string operators (+ and -) in order from most important to least important. In the example that follows, EMPLOYEE.DBF is indexed on Last Name within Job Code for each Department, resulting in the sequence shown in Figure 9-1. Department is the primary (most important) field; Last Name is least important.

Table 9-1 Alphabetical Employee List by Job Code and Department

DEPT.	JOB CODE	NAME
16	p3	Rowland, Paul
	p4	Smyth, Gail
	p9	Freitag, Jean
38	p3	Destry, Ralph
54	ml	Green, Terry
	p3	Brown, John
	s8	Peters, Alice
75	d8	Harris, Richard
	s8	Alazar, Pat
		Schaller, Paula
89	p3	Gilbert, Diane
	p8	Embry, Albert
	s8	Howser, Peter

The following screen demonstrates how this is done.



```
. use employee
. index on dept:num+job:code+last:name to concatx
00017 RECORDS INDEXED
. use employee index concatx
. list dept:num job:code last:name
00014  16 p3 Rowland
00017  16 p4 Smyth
00006  16 p9 Freitag
00016  38 d8 Shaffer
00004  38 p3 Destry
00008  54 m1 Green
00003  54 p3 Brown
00009  54 s8 Green
00013  54 s8 Peters
00012  66 p9 Inders
00010  75 d8 Harris
00001  75 s8 Alazar
00015  75 s8 Schaller
00007  89 p3 Gilbert
00002  89 p8 Berger
00005  89 p8 Embry
00011  89 s8 Howser
```

Concatenating fields accomplishes the same thing as multiple SORTs, and it only requires one command.

NOTE

All fields in concatenated keys must be character type. Numeric fields must first be converted to character type using the STRING function (see Chapter 15) . For example, the EMPLOYEE file can be indexed on employee number (NUMBER) within department (DEPT:NUM).

USING INDEXED FILES

A data base file should not be indexed unless an application that uses the file benefits from the feature. Indexes are useful when one file must be ordered on more than one key sequence. Indexing also allows very rapid location of data records when the key is specified in the FIND command.

An indexed data base file functions like a sequential, non-indexed file when it is opened with the form of the USE command (USE <file>) already introduced. To take advantage of the speed built into an indexed file, you must associate an index with the file, as follows.

USE <file> INDEX <index file list>

The first file named in the index file list is called the *master index*. It determines the sequence of records and is the reference for any FIND commands. The other indexes in the list are not used for sequencing, but are automatically updated by certain commands as explained later in this chapter. dBASE II automatically maintains up to seven index files for the data base file in USE. The file names must be separated by commas.

The master index can be changed by the following command.

SET INDEX TO <new index file list>

The SET command closes all currently open index files and then opens the index files named in the list. The data base file is not closed and its current record pointer setting is not changed. However, the relationship between the master index file and the data base file is lost. A FIND command or GOTO must be issued to reset the index pointer of the new master index file before any commands that have a NEXT phrase are issued.

A SET INDEX TO command with no index file list closes all index files for the data base file in USE. The USE file then functions as a sequential file.

```
. use employee
. set index to concatx, deptndx
. go top
. display
00014  Rowland      Paul      13 709 Key St    Houston    TX 77007 16 p3
15750.00 A
. skip 3
RECORD: 00016
. display
00016  Shaffer      Peter      8 43 Clinton Av Montclair  NJ 07042 38 d8
17900.00 A
. set index to deptndx
. go top
. display
00006  Freitag      Jean       9 854 Moose Blv Houston    TX 77006 16 p9
2775.00 A
. skip 3
RECORD: 00004
. display
00004  Destry       Ralph      3 234 Mahogony  Deerfield  FL 33441 38 p3
15575.00 A
. set index to
. go top
. display
00001  Alazar       Pat        1 123 Crater    Everett    WA 98206 75 s8
12500.00 A
. skip 3
RECORD: 00004
. display
00004  Destry       Ralph      3 234 Mahogony  Deerfield  FL 33441 38 p3
15575.00 A
```

Finding Records in an Indexed File

To locate one or more records that have a particular key value, use the FIND command. The system searches the index file for the first record whose key field matches the value specified in the FIND command. It then sets the current record pointer in the data base file to the appropriate record.

FIND operates only on data base files that have been previously indexed and that are in USE with an index file.

The FIND command has two forms:

FIND <char string>
FIND '<char string>'

In most cases, use the first form of the command, without quotes. Use the second form of the command, with quote marks enclosing the character string, when the key field has leading blanks and is a character type field. Also use quotes with concatenated keys that have leading or trailing blanks. Include the exact number of blanks in the character string.

FINDING CHARACTER STRING KEYS

If the data base file was indexed using a character string expression as the key, then FIND compares the key values in the index file with as much of the key as is specified in the command. The record found is the first one whose key matches the character string. For example, a record with the key "SMITH, JOHN" could be found by the statement FIND SMI provided that there are no other keys starting with SMI preceeding SMITH, JOHN in the index. The command always finds only the first record whose key is the same as the character string. Even if the record pointer is moved down further in the file, a subsequent FIND on the same key points to the first record with that key.

Comparison of character strings in dBASE II proceeds only as far as the length of the second argument. To require an exact string comparison, issue the command SET EXACT ON. This feature requires that all characters except trailing blanks are identical for a true match to be indicated.

FINDING NUMERIC KEYS

If the index was created with a numeric key, then the found record will be the first record whose key is arithmetically equal to the object of the FIND.

FINDING CONCATENATED KEYS

FIND works with a file indexed on concatenated keys when the key values are placed within quotes and the full fields, including leading and trailing blanks are included.

```
. use employee index deptndx
. find 75
. display
00001  Alazar       Pat        1 123 Crater    Everett    WA 98206 75 s8
12500.00 A
. find 89
. display
00002  Berger       Mary       6 10 Kearney Rd Needham    MA 02194 89 p8
30000.00 A
. find 16
. display
00006  Freitag      Jean       9 854 Moose Blv Houston    TX 77006 16 p9
2775.00 A
. set index to concatx
. find '75s5Schaller      '
NO FIND
. find '38p3Destry      '
. display
00004  Destry       Ralph      3 234 Mahogony  Deerfield  FL 33441 38 p3
15575.00 A
```

Using the Found Record

Once a record in a data base file has been selected with the FIND command, it can be processed just like any other data base record. It can be interrogated, altered, displayed, and so on. dBASE II commands that reposition the record pointer process the found record first and proceed to the next record in *key sequence*.

If there is no record in the index file whose key matches the character string in the FIND command, the message "NO FIND" is displayed on the screen and the record number function, #, returns the value zero.

Finding More

To access additional records with the same key, use either the SKIP or LOCATE FOR <exp> command. SKIP does not know when there is no longer a match. LOCATE finds additional matches provided <exp> equals the key value.

The example that follows uses the EMPLOYEE file indexed on dept:num.

```
. use employee index deptndx
. find 89
. display
00002  Berger        Mary        6 10 Kearney Rd Needham     MA 02194 89 p8
30000.00 A
. locate next 3
RECORD: 00002
. display
00002  Berger        Mary        6 10 Kearney Rd Needham     MA 02194 89 p8
30000.00 A
. continue
RECORD: 00005
. display
00005  Embry         Albert      2 345 Sage Ave  Palo Alto   CA 94303 89 p8
22200.00 A
. continue
RECORD: 00007
. display
00007  Gilbert       Diane      14 280 Cactus Wy Las Cruces  NM 88001 89 p3
24500.00 A
. continue
END OF LOCATE SCOPE
. store dept:num to sdept
89
. go top
. locate for dept:num = sdept
RECORD: 00002
. continue
RECORD: 00005
. continue
RECORD: 00007
. continue
RECORD: 00011
. continue
END OF FILE ENCOUNTERED
```

Command Operations with Indexed Files

Some commands operate with indexed files differently than with sequential files. This section details the effects of using one or more indexes simultaneously with a data base file.

Positioning Commands

Commands that operate by record number (GOTO, SKIP, EDIT, and the RECORD <n> and NEXT <n> options for scope) move through the index file, not the actual data base file. That is, the current record pointer is positioned in the index file, and it uses information contained there to access the indicated data record. Moving between records yields the next or previous indexed record relative to the current record pointer.

Test the effect with GOTO, SKIP, or any other commands that accept the scope phrase. Then try the EDIT and BROWSE commands. In edit mode, **CTRL R** and **CTRL C** also use the index file for positioning.

Commands that Change File Contents

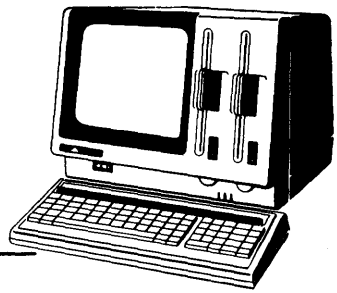
APPEND automatically updates all files specified in <index list> in the USE or SET INDEX TO command for new data records unless the records are blank. Records appended with the BLANK option are not indexed until they are supplied with some data. Any index files associated with a data base file and not in USE when the file is updated are not automatically updated and must be indexed again.

INSERTs into an indexed file are identical to APPENDs for the Master Index file.

The EDIT, BROWSE, and REPLACE commands delete a record from the index file and reenter a new index entry if the key field is altered for any index file except the Master Index. Do not edit the key field of the Master Index. The index files must be in USE with the data base file or called with a SET INDEX TO <index list> command prior to any modifications.

In an indexed file, the key should not be REPLACEd with NEXT <n> as the scope phrase.

An indexed file can be packed with the PACK command and the data base file, as well as the Master Index file, will be properly adjusted. For large indexed files, PACKing the file without the index and then reindexing is faster than PACKing with the index. If a data base file has more than one index associated with it, the recommended technique is to PACK the file without any indexes and then index the file again.



Chapter 10

Reporting The Contents Of Data Bases

At some point you will need to summarize and quantify what is in the data base. You may need formal reports or statistics about the data. The REPORT feature was introduced in Chapter 3. It is fully explained in this chapter, along with other commands that allow you to manipulate, reorganize, and format data to present it in the manner you require.

SUMMARIZING THE RECORD CONTENTS

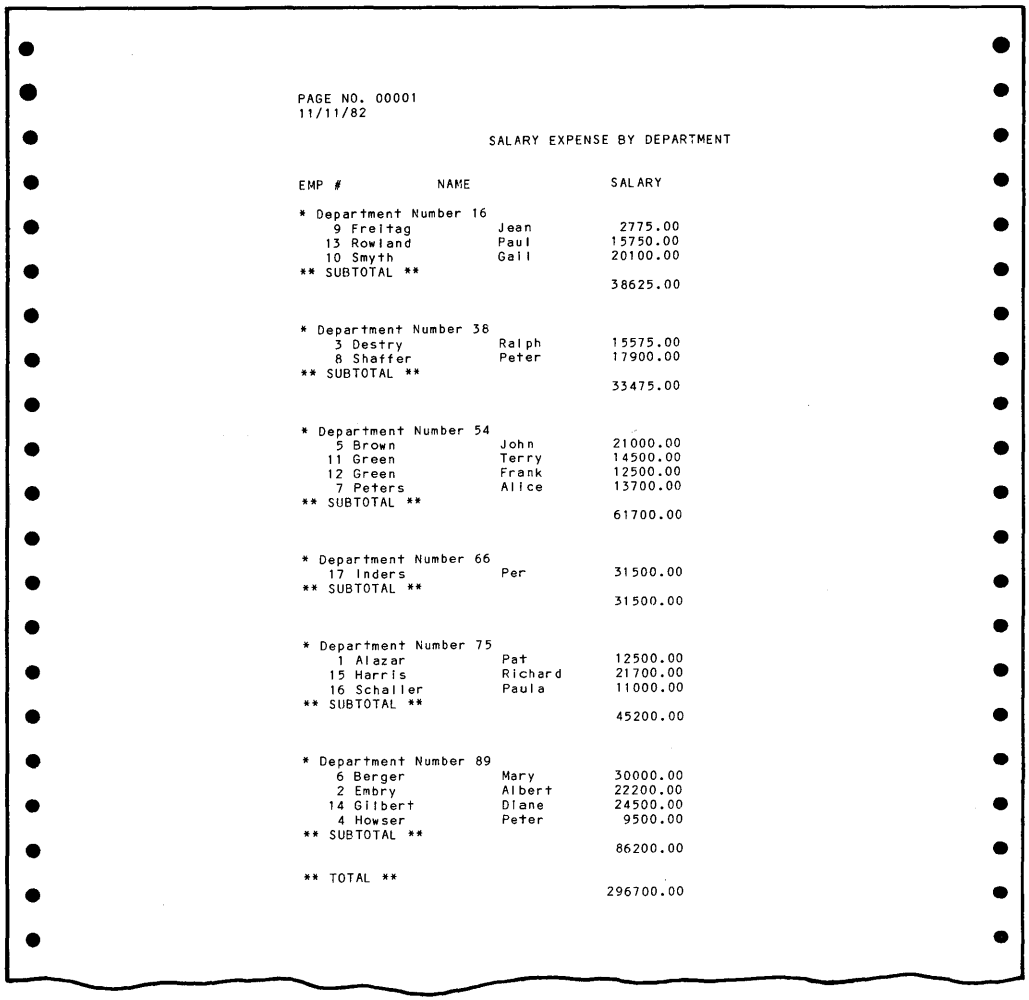
In most applications, you are likely to need formatted listings of data base contents and, perhaps, other information derived from the data. The REPORT command prepares reports and displays them on the screen or prints them in "hard copy" on paper. It can also be used to format tables of data.

Creating a Report Form File

Defining a report is similar to defining a file. The system leads you through a dialog in which you describe the layout and contents of the report. Reports can have titled columns, totaled numeric fields, and displayed expressions involving data fields, memory variables, and constants.

The following screen illustrates the dialog that produced the SALARY EXPENSE BY DEPARTMENT report, Figure 10-1.

```
. use employee Index deptndx
. report
ENTER REPORT FORM NAME: salaries
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH
PAGE HEADING? (Y/N) y
ENTER PAGE HEADING: SALARY EXPENSE BY DEPARTMENT
DOUBLE SPACE REPORT? (Y/N) n
ARE TOTALS REQUIRED? (Y/N) y
SUBTOTALS IN REPORT? (Y/N) y
ENTER SUBTOTALS FIELD: dept:num
SUMMARY REPORT ONLY? (Y/N) n
EJECT PAGE AFTER SUBTOTALS? (Y/N) n
ENTER SUBTOTAL HEADING: Department Number
COL     WIDTH,CONTENTS
001     5,number
ENTER HEADING: EMP #
002     16,last:name
ENTER HEADING:          NAME
003     10,first:name
ENTER HEADING:
004     10,salary
ENTER HEADING: SALARY
ARE TOTALS REQUIRED? (Y/N) y
005
```



PAGE NO. 00001
11/11/82

SALARY EXPENSE BY DEPARTMENT

EMP #	NAME		SALARY
* Department Number 16			
9	Freitag	Jean	2775.00
13	Rowland	Paul	15750.00
10	Smyth	Gail	20100.00
** SUBTOTAL **			38625.00
* Department Number 38			
3	Destry	Ralph	15575.00
8	Shaffer	Peter	17900.00
** SUBTOTAL **			33475.00
* Department Number 54			
5	Brown	John	21000.00
11	Green	Terry	14500.00
12	Green	Frank	12500.00
7	Peters	Alice	13700.00
** SUBTOTAL **			61700.00
* Department Number 66			
17	Inders	Per	31500.00
** SUBTOTAL **			31500.00
* Department Number 75			
1	Alazar	Pat	12500.00
15	Harris	Richard	21700.00
16	Schaller	Paula	11000.00
** SUBTOTAL **			45200.00
* Department Number 89			
6	Berger	Mary	30000.00
2	Embry	Albert	22200.00
14	Gilbert	Diane	24500.00
4	Howser	Peter	9500.00
** SUBTOTAL **			86200.00
** TOTAL **			296700.00

Figure 10-1 Salary Expense by Department Report

The dialog is described in the next section of this chapter. To see how the REPORT command works, enter the report description as it appears in the preceding example or create another report by following along with the rules listed here.

PREPARATION

Plan your reports as carefully as you plan your data base structures. Each time you create a report, follow this procedure.

1. Define the report. The example shows a report with the following features:

   - Report heading is "SALARY EXPENSE BY DEPARTMENT".
   - Employees are grouped by department and salary is subtotaled within department.
   - Department subtotals are printed after the detail records for each employee. Detail records list employee name (last name first), number and salary.

   If you are defining a new report rather than copying the example, take a piece of paper and plan what the report should look like. Include headers and field names and see how much space to allow for each column. Decide what fields should be subtotaled and totaled.

2. Issue all SET commands.

   - If you want a printed copy of the report, turn the printer on. Then press **PRINT** on the numeric keypad, issue the command SET PRINT ON, or use the phrase "TO PRINT" in the REPORT command.
   - If you are printing on single sheets of paper, enter the command **SET EJECT OFF** to suppress the initial form feed.

3. USE the file that contains the data (with the INDEX option if necessary). REPORT requires an open file.

4. Issue the REPORT command. In the example, it is presented without modifiers. This creates a new report form file and initiates the report formatting dialog.

REPORT FORMATTING DIALOG

This section describes the formatting dialog. The system prompts are printed in uppercase, just as they appear on the screen. Following each prompt are definitions, restrictions, and descriptions of the system's actions based on your input.

ENTER REPORT FORM NAME:

- Follow standard eight-character file name rules (see Chapter 5).
- Enter <file>, including the drive specifier if necessary.
- The system automatically assigns the ".FRM" extension.

ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH

- Standard report layout is:
  LEFT MARGIN, M=8
  LINES/PAGE, L=56
  PAGE WIDTH, W=80
- Page width controls only the centering of the PAGE HEADING. The "Employees by Department" report in Chapter 3 could be cleaned up either by centering the columns under the heading or placing the heading over the data.
- Enter options(s), separated by commas, or press **RETURN** to accept the defaults.

PAGE HEADING? (Y/N)

- Enter **Y** If you want a heading printed at the top of each page; otherwise enter **N**.

ENTER PAGE HEADING:

- This prompt appears only if you answered **Y** to the previous question.
- The limit is 254 characters.
- Enter text for the heading without quotation marks.
- The semicolon (;) may be used to mark a line break. The display is continued on the next line.

DOUBLE SPACE REPORT? (Y/N)

- Enter **Y** or **N**.

ARE TOTALS REQUIRED? (Y/N)

- REPORT can total some or all numeric fields.
- Enter **N** if you do not want any totals.
- If **Y** is entered, you have the option of printing the total for each numeric field in the report. The system prompts you as you enter descriptions of numeric fields.

SUBTOTALS IN REPORT? (Y/N)

- This question appears only if the report requires totals.
- REPORT can subtotal on one field only.
- Enter **Y** or **N**.

The next four prompts refer to subtotaling. They appear only when subtotals are required.

ENTER SUBTOTALS FIELD:

- The file must be indexed on, or sorted by, this field.
- Enter <field>.

SUMMARY REPORT ONLY? (Y/N)

- If **Y** is entered, only the headings, column subtotals, and report totals are produced.
- If **N** is entered, headings, detail records, subtotals, and report totals are produced.

EJECT PAGE AFTER SUBTOTALS (Y/N)

- Enter **Y** or **N**.

ENTER SUBTOTAL HEADING:

- Enter the character string to be used as the heading.
- The character string is concatenated with the subtotal field and appears preceeding any detail records in the format

  * <character string> <field>

The next prompts accept the description of each column in the report. Enter the width and contents of each field, or press **RETURN** instead of entering a width to terminate the report description process.

```
COL   WIDTH, CONTENTS
00n
```

COL

- Column number is generated by the computer.
- Reports may have up to 24 columns.
- Number is used for identification only. It does not appear in the report.

WIDTH

- This is the number of positions allotted to the column.
- There is not necessarily a direct relationship between the column width and the width of a field in a file.
- If the data (header or contents) are less than the width, numeric values are right-justified and alphanumeric values are left-justified.
- If the data are greater than the width, the data wrap around to more than one line. For example, an 80-character field wraps to two lines in a 50-character column.
- dBASE II inserts one space between columns. To allow more space between columns, add extra characters to the width or concatenate spaces to data in the contents field.

CONTENTS

- Describe the source of the data. Acceptable entries include fields (FIRST:NAME), memory variables, literals ("ABC"), and expressions (SALARY * 1.15).
- Concatenated strings may be used (DEPT:NUM + ".....").

ENTER HEADING:

Enter the literal data to be displayed above the column.

- Heading must fit in the WIDTH or it wraps around.
- Special symbols can be used as follows:

  ; Semicolons embedded in headings and strings mark line breaks.

  > As the first character, this symbol indicates that the heading should be right-justified in the column.

  < As the first character, this symbol indicates that the heading should be left-justified in the column.

ARE TOTALS REQUIRED? (Y/N)

- This question appears only for numeric fields and only if you requested totals in the overall report description.
- Enter **Y** or **N**.

When all columns in the report have been defined, press **RETURN** at the next field number prompt. dBASE II immediately starts printing the report. It will process the entire data base if you let it. To stop the report display, press **ESC.**

HINT

It is a good idea to keep a printed copy of report form file descriptions as you create them since it is difficult to access them later. (You cannot simply USE and LIST a report form file.) To print the file, use SET PRINT ON (or press **PRINT**) before issuing the REPORT command. From that point on, until you enter SET PRINT OFF or press **PRINT** again, everything that appears on the screen is also printed.

Using Report Form Files

Once a report form has been described, it is saved in a file with the extension ".FRM". Report form file names may be listed with the command LIST FILE LIKE *.FRM. These files can be modified using a word processor or text editor, but it is not recommended that you do so. It is usually easier to define a new report form file.

You can use the same report form to generate different reports by displaying the contents of different data bases, using phrases to report selectively, and using SET commands to exercise options at run time.

- Any data base file can be USEd as input to REPORT if it contains all the fields used in the report form file.

- The complete form of the REPORT command is:

  REPORT [FORM <form>] [<scope>] [FOR <exp>] [TO PRINT] [PLAIN]

- The optional phrases may be used to control reporting.
  - FORM is required to use an existing form file.
  - <scope> defaults to all when not specified.
  - FOR allows only the information which meets the conditions of the expression to be reported.
  - TO PRINT sends the report to the printer as well as the screen. The printer must be turned on. This phrase may be used instead of issuing the SET PRINT ON command or pressing **PRINT**.
  - PLAIN suppresses the printing of page numbers and date at the top of each page. The page heading is printed only at the beginning of the report.
- SET commands issued at run time allow you to exercise additional controls.
  - The date appears under the page number. The system defaults to the date entered at sign-on. It can be changed to another date or spaces by the SET DATE TO command.
  - An additional heading may be generated for each run with the command SET HEADING TO <char string>.

    The character string limit is 60 characters. Do not use quotation marks in the character string. This heading must be set each time a new dBASE II session is initiated. (The standard report heading entering through the dialog is saved in the report form file).
- The report medium is controlled by the SET PRINT ON command, the **PRINT** key, or the TO PRINT phrase in the REPORT command.
- Before printing anything, REPORT ejects the page in the printer. To suppress this, enter SET EJECT OFF. If you are using single sheets of paper in the printer, be sure to use this command.

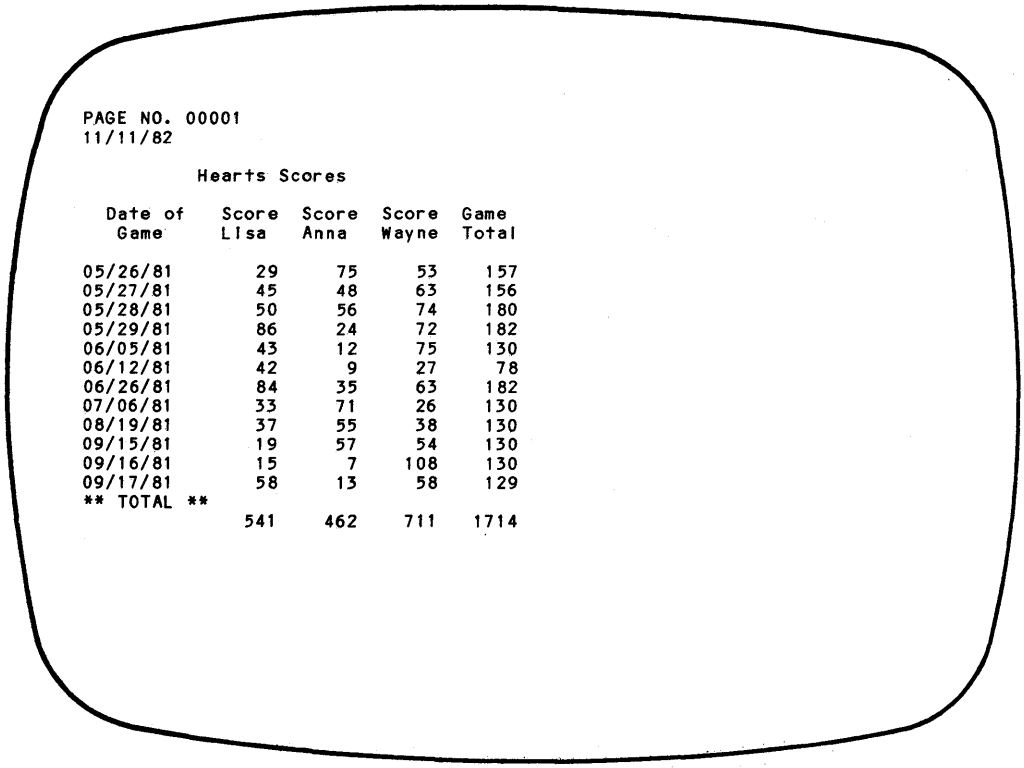
The screens that follow demonstrate some of the run-time options available with the REPORT command. They use a data base of scores for three people in an on-going card game.

```
. use cards
. list structure
STRUCTURE FOR FILE:  CARDS.DBF
NUMBER OF RECORDS:   00012
DATE OF LAST UPDATE: 11/11/82
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH   DEC
001       DATE          C    008
002       LISA          N    003
003       ANNA          N    003
004       WAYNE         N    003
** TOTAL **              00018
. list
00001  05/26/81   29   75   53
00002  05/27/81   45   48   63
00003  05/28/81   50   56   74
00004  05/29/81   86   24   72
00005  06/05/81   43   12   75
00006  06/12/81   42    9   27
00007  06/26/81   84   35   63
00008  07/06/81   33   71   26
00009  08/19/81   37   55   38
00010  09/15/81   19   57   54
00011  09/16/81   15    7  108
00012  09/17/81   58   13   58
```

Column headings can be printed on two lines by using the semicolon to mark the line break. Column contents can be fields names and expressions.

```
. use cards
. report
ENTER REPORT FORM NAME: cards
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH w=40
PAGE HEADING? (Y/N) y
ENTER PAGE HEADING: Hearts Scores
DOUBLE SPACE REPORT? (Y/N) n
ARE TOTALS REQUIRED? (Y/N) y
SUBTOTALS IN REPORT? (Y/N) n
COL      WIDTH,CONTENTS
001      10,date
ENTER HEADING: Date of;Game
002      6,lisa
ENTER HEADING: Score;Lisa
ARE TOTALS REQUIRED? (Y/N) y
003      6,anna
ENTER HEADING: Score;Anna
ARE TOTALS REQUIRED? (Y/N) y
004      6,wayne
ENTER HEADING: Score;Wayne
ARE TOTALS REQUIRED? (Y/N) y
005      6,lisa+anna+wayne
ENTER HEADING: Game;Total
ARE TOTALS REQUIRED? (Y/N) y
006
```

The "Hearts Scores" format is saved in a file called CARDS.FRM. The report can be produced at any time with the REPORT command.



PAGE NO. 00001
11/11/82

Hearts Scores

Date of Game	Score Lisa	Score Anna	Score Wayne	Game Total
05/26/81	29	75	53	157
05/27/81	45	48	63	156
05/28/81	50	56	74	180
05/29/81	86	24	72	182
06/05/81	43	12	75	130
06/12/81	42	9	27	78
06/26/81	84	35	63	182
07/06/81	33	71	26	130
08/19/81	37	55	38	130
09/15/81	19	57	54	130
09/16/81	15	7	108	130
09/17/81	58	13	58	129
** TOTAL **	541	462	711	1714

An additional heading can be added to the report during the dBASE II session. The PLAIN option and conditional phrases can also be used to change the contents and format of the report.

```
. use cards
. set heading to MAY
. report form cards next 4

        PAGE NO. 00001       MAY
        11/11/82

                  Hearts Scores

          Date of    Score  Score  Score  Game
           Game      Lisa   Anna   Wayne  Total

        05/26/81        29     75     53    157
        05/27/81        45     48     63    156
        05/28/81        50     56     74    180
        05/29/81        86     24     72    182
        ** TOTAL **
                       210    203    262    675

. goto top
. report form cards for wayne < 50 plain

                  Hearts Scores

          Date of    Score  Score  Score  Game
           Game      Lisa   Anna   Wayne  Total

        06/12/81        42      9     27     78
        07/06/81        33     71     26    130
        08/19/81        37     55     38    130
        ** TOTAL **
                       112    135     91    338
```

PREPARING TABLES WITH REPORT

PLAIN is an option with the REPORT command. It allows you to create reports and tables that can be inserted into reports generated by a word processor. You can even use the PLAIN option to surround the report with other calculations or text.

QUANTIFYING THE FILE CONTENTS

In some applications, you don't need to see the actual records, but want to know how many meet certain conditions, or what the total is for some specified condition. (How many widgets do we have in stock? How many are on back order? What is the total of our A/P?)

For counting use the COUNT command.

COUNT [<scope>] [FOR <exp>] [TO <memvar list>]

This command can be used with none, some or all of the modifiers. Unqualified, it counts all the records in the data base, including those marked for deletion.

- <scope> can be limited to one or a specified number or records. The default is ALL.
- The FOR phrase can be any simple or complex expression.
- The result can be stored in a memory variable (see Chapter 12) when the TO phrase is used. If memory variable does not already exist, it is created when the command is executed.

```
. use employee
. count
COUNT = 00017
. count for dept:num = '89'
COUNT = 00004
. count for "G" $ last:name
COUNT = 00003
. count for job:code = 'p3' .and. dept:num = '89'
COUNT = 00001
```

To total data base field values, use the SUM command.

SUM <field list> [<scope>] [FOR <exp>] [TO <memvar list>]

This command computes and displays the totals for all named fields in records that meet the conditions in the FOR phrase.

- You can list up to five numeric fields to total in the data base in USE.
- If more than one field or expression is to be totaled, the field names must be separated by commas.
- <scope> defaults to ALL; other values may be used to limit the records included in the scope. Records marked for deletion are bypassed.
- The FOR phrase can be a simple or complex expression.
- The results can be stored in memory variables, with one sum in each variable named. There is a one-to-one correspondence, based on position in the lists, between the summed fields and the memory variables.

The following screen shows examples of the SUM command formats.

```
. use employee
. sum salary
 296700.00
. sum salary for dept:num = '75'
 45200.00
. sum salary for job:code = 'p'
 183400.00
. sum salary for job:code = 'p' .or. job:code = 'm'
 197900.00
. sum dept:num
sum DEPT:NUM
CORRECT AND RETRY (Y/N)? N
```

SUMMARIZING DATA AND ELIMINATING DETAILS

The TOTAL command is similar to the subtotal capability in the REPORT command. It sums data fields, but instead of printing them out, it stores the subtotals in a data base file that is a summary of the source file.

TOTAL ON <key> TO <file> [FIELD <field list>] [FOR <exp>]

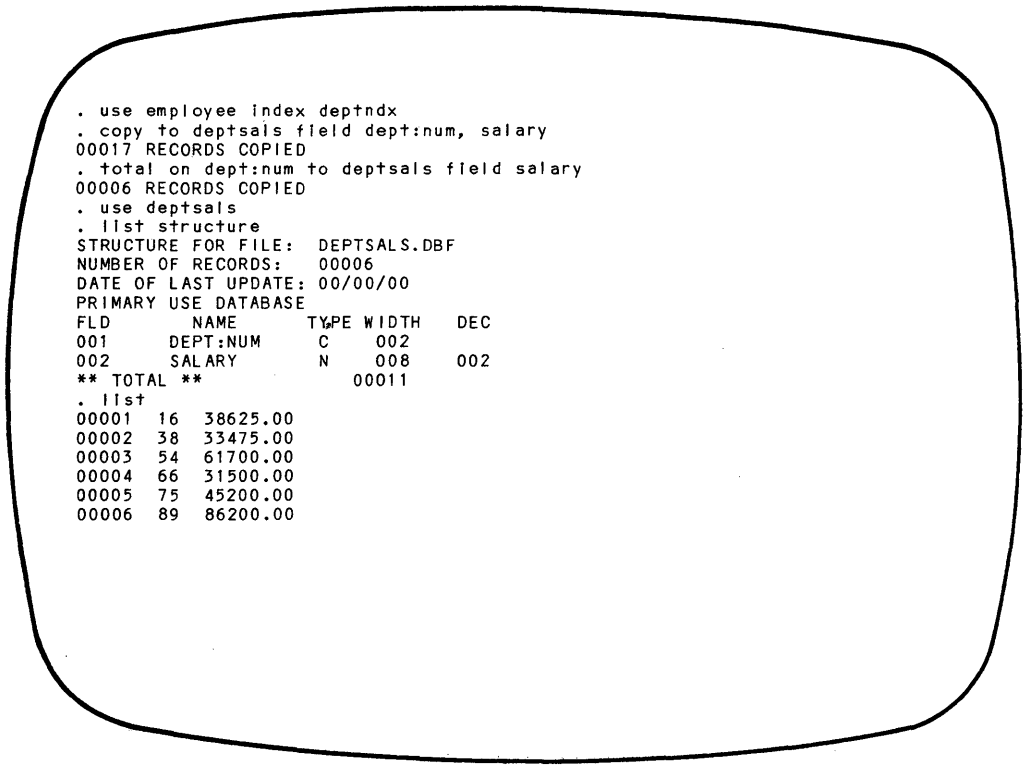
- The USE data base file must be presorted or indexed on <key>.
- <file> may be a new or existing data base. If it already exists, its structure is left intact.
- If <file> did not exist prior to this TOTAL command, the structure is copied from the USE file.
- If the FIELD phrase is included, only the named fields are totaled. If the phrase is not present, all numeric fields are totaled.

```
. use employee index deptndx
. total on dept:num to deptsals field salary
00006 RECORDS COPIED
. use deptsals
. list structure
STRUCTURE FOR FILE:  DEPTSALS.DBF
NUMBER OF RECORDS:   00006
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH   DEC
001      LAST:NAME     C    012
002      FIRST:NAME    C    007
003      NUMBER        N    003
004      STREET        C    013
005      CITY          C    010
006      STATE         C    002
007      ZIP:CODE      C    005
008      DEPT:NUM      C    002
009      JOB:CODE      C    002
010      SALARY        N    008    002
011      ACT:STAT      C    001
** TOTAL **             00066
. list
00001  Freitag      Jean        9 854 Moose Blv Houston    TX
77006 16 p9  38625.00 A
00002  Destry       Ralph       3 234 Mahogony  Deerfield  FL
33441 38 p3  33475.00 A
00003  Brown        John        5 456 Minnow Pl Burlington MA
01730 54 p3  61700.00 A
00004  Inders       Per        17 321 Sawtelle  Tuscon     AZ
85702 66 p9  31500.00 A
00005  Alazar       Pat         1 123 Crater    Everett    WA
98206 75 s8  45200.00 A
00006  Berger       Mary        6 10 Kearney Rd Needham    MA
02194 89 p8  86200.00 A
```

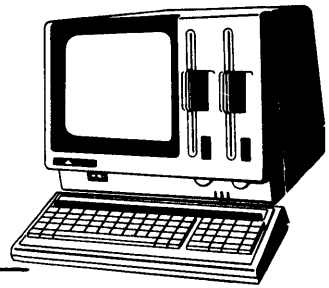
This command is most selective when the TO file, consisting of only the fields used in the TOTAL command, exists and the FIELD phrase is used. The next screen demonstrates this. First, the TO data base is created using COPY TO <file> FIELD <fieldl> [,<field2>... [,<fieldn>] where:

<file> = "TO" file for TOTAL command
<fieldl> = <key> in TOTAL command
<field2>...<fieldn> = FIELDS in TOTAL command.

Then, the TOTAL command is issued.



```
. use employee index deptndx
. copy to deptsals field dept:num, salary
00017 RECORDS COPIED
. total on dept:num to deptsals field salary
00006 RECORDS COPIED
. use deptsals
. list structure
STRUCTURE FOR FILE:  DEPTSALS.DBF
NUMBER OF RECORDS:   00006
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH   DEC
001      DEPT:NUM     C    002
002      SALARY       N    008     002
** TOTAL **              00011
. list
00001   16   38625.00
00002   38   33475.00
00003   54   61700.00
00004   66   31500.00
00005   75   45200.00
00006   89   86200.00
```



Chapter 11

Using The Printer

You will probably use the printer most often to print hard copies of reports. However, there are times when you need printed copies of listings, command files, and screen displays. This chapter summarizes the commands for using the printer. Before using any print command, turn the printer on and align paper in it.

PRINTING REPORTS

The easiest way to print a report is to use the TO PRINT option in the REPORT command. This prints the named report, and only that report.

REPORT automatically performs a page eject before printing any report. If you are using single sheets of paper, suppress the page eject by entering **SET EJECT OFF** before the REPORT command. If you are using continuous paper, the SET command is optional.

INTERACTIVE PRINT CONTROL

There are three ways to control printing from the console.

The **PRINT** key is located in the top row of the numeric keypad on the right side of the keyboard. It toggles the printer on and off. When you start a dBASE II session, the print function is set to off. To print, press **PRINT**. From then on, until you press **PRINT** again, everything that you enter and most of the dBASE II commands and prompts that appear on the screen are also printed.

Pressing **CTRL P** also toggles the printer on and off. **CTRL P** operates exactly like **PRINT**.

The **SET PRINT ON** and **SET PRINT OFF** commands can be used to control printing. They may be entered at the dot prompt.

PRINTING COMMAND FILES

It is a good idea to keep hard copies of your dBASE II programs and command files. Use the listings when you are writing and debugging new routines. Save printed copies of working command files for backup and reference.

dBASE II does not have a capability to print the contents of command files. To print these files, you must exit from dBASE II and return to the CP/M-86 "A>" prompt. The CP/M-86 program for the APC includes a command called TYPE that lists and prints CP/M files. Use the following procedure to print command files.

1. Exit from dBASE II. The system displays the CP/M-86 "A>" prompt.
2. Turn on the printer and align paper in it.
3. Press either **PRINT** or **CTRL P** to set print function on.
4. Enter the TYPE command, specifying the full name of the file.
   TYPE [<disk drive>:] <file name>. <type>

   The system responds by printing the contents of the command file.
5. When the file listing is completed, press **PRINT** or **CTRL P** to turn off the printer.

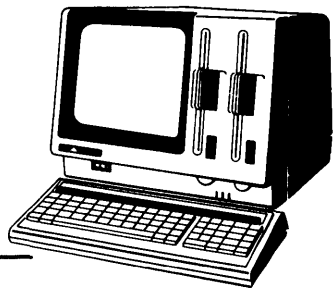
NOTE

See the *CP/M-86 System User's Guide* for more information about using the TYPE command.

PRINTING FROM COMMAND FILES

The SET PRINT command just described can be used in command files to control printing.

SET FORMAT TO PRINT sends the outpout of @ commands (see Chapter 18) to the printer instead of the display screen.



Chapter 12

Generating Reusable Variables And Instructions

MEMORY VARIABLES

dBASE II reserves a part of its memory for storing variables that you name. You might want to think of this area as a series of pigeon-holes available for you to tuck data into while you are working out a problem. Memory variables are not fields in a data base; they exist simply to hold data temporarily.

You can use a memory variable to hold a value you need to use in different commands and expressions. You can then refer to the memory variable instead of entering the same data repetitively. For example, in one dBASE II session, you might "tuck" the day's prime rate into a pigeon-hole (variable) named RATE. During the session, you could get it by asking for RATE. You could then place the value into a prime rate field in any file, or use it in calculations, without having to reenter it.

dBASE II allocates enough memory to store up to 64 memory variables, each with a maximum length of 254 characters, up to a maximum total of 1536 characters for all the variables.

Creating Memory Variables

The STORE command creates a memory variable and stores a value in it at the same time. The complete form of the command is:

STORE <var/constant/exp> TO <memvar>

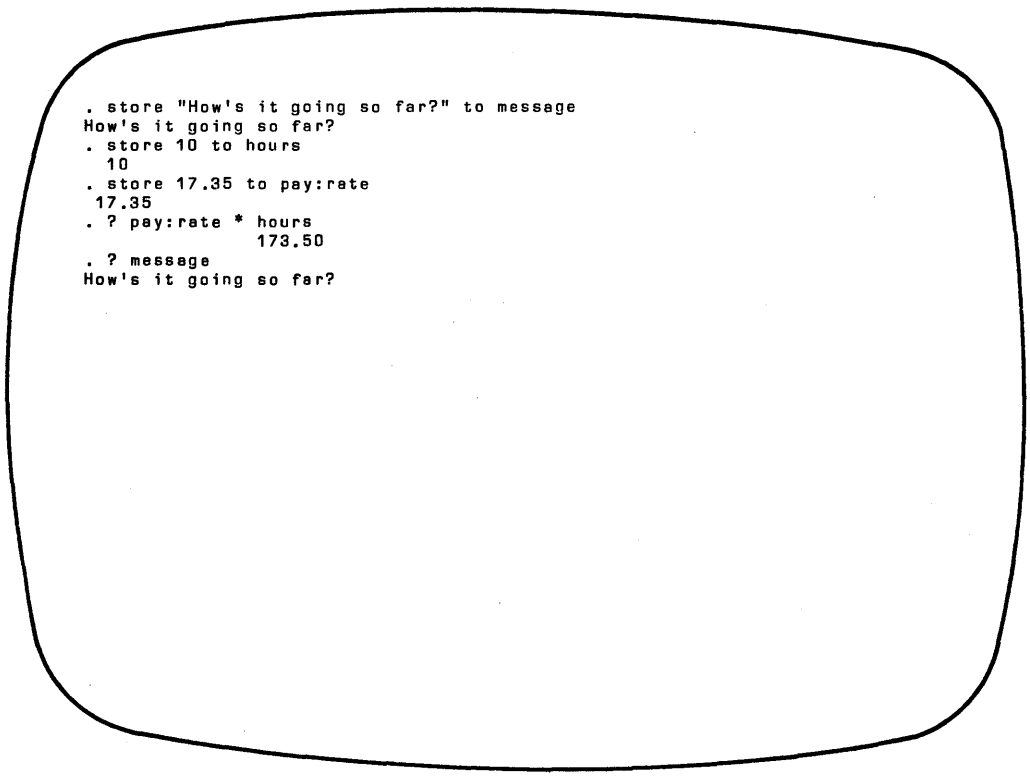
The stored value can be a variable or constant or the result of an expression. If the memory variable did not exist before the command was issued, dBASE II creates it automatically. Memory variable names can be any legal dBASE II identifier.

NOTE

If you use only nine characters for data base field names, when you want to use the name as a memory variable, you can do so by putting an "M" in front of it. What it stands for will be clearer than if the variable were assigned a completely different name.

To see how the STORE command works with character and numeric data and expressions, enter the following commands at the prompt.

store "How's it going so far?" to message
store 10 to hours
store 17.35 to pay:rate
?pay:rate * hours
? message



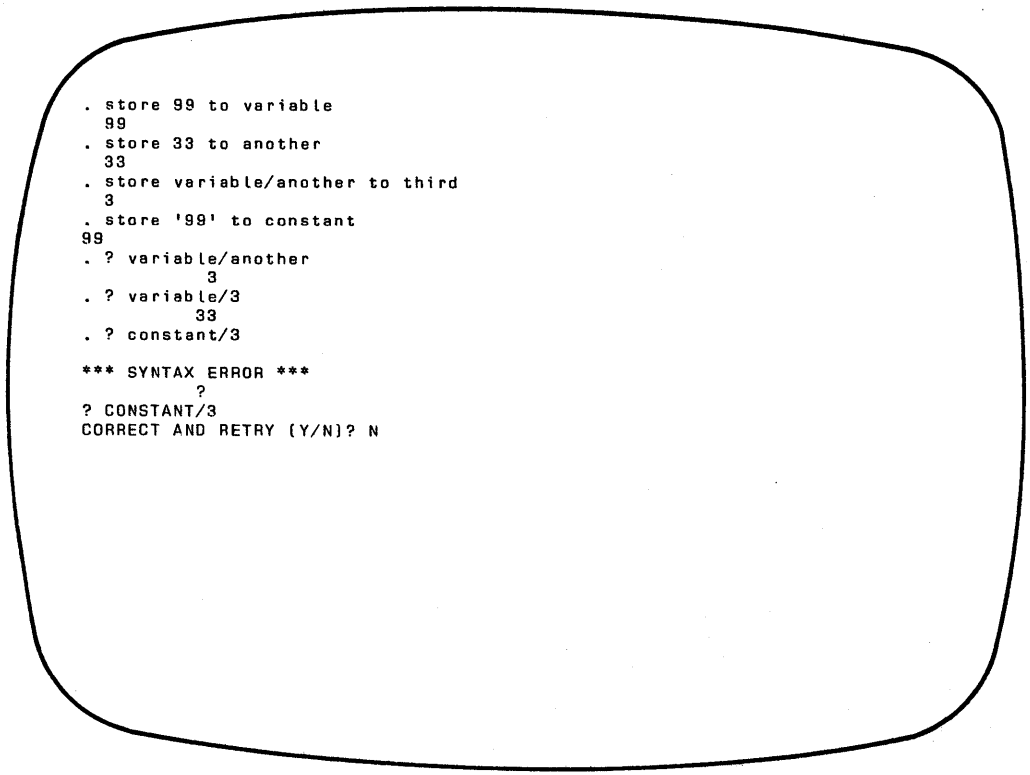
```
. store "How's it going so far?" to message
How's it going so far?
. store 10 to hours
  10
. store 17.35 to pay:rate
 17.35
. ? pay:rate * hours
              173.50
. ? message
How's it going so far?
```

Notice the double quotes around the character string, a constant, in the first command. They are necessary because a single quote is used as an apostrophe inside the string.

If the differences between constant and variable data are not clear, try experimenting with and without the quotes. To start off, enter the following commands.

store 99 to variable
store 33 to another
store variable/another to third
store '99' to constant
? variable/another
? variable/3
? constant/3

Your screen will look like the display that follows.



```
. store 99 to variable
  99
. store 33 to another
  33
. store variable/another to third
  3
. store '99' to constant
99
. ? variable/another
          3
. ? variable/3
         33
. ? constant/3

*** SYNTAX ERROR ***
          ?
? CONSTANT/3
CORRECT AND RETRY (Y/N)? N
```

Storing a value into a memory variable automatically tells the system what the data type is. From then on, you cannot mix data types. The last example demonstrates the results of trying to divide a character string by a number.

Rules for Character String Memory Variables

Character strings that appear in expressions must be enclosed in matching single or double quote marks or square brackets.

Character strings may contain any of the printable characters, including the space.

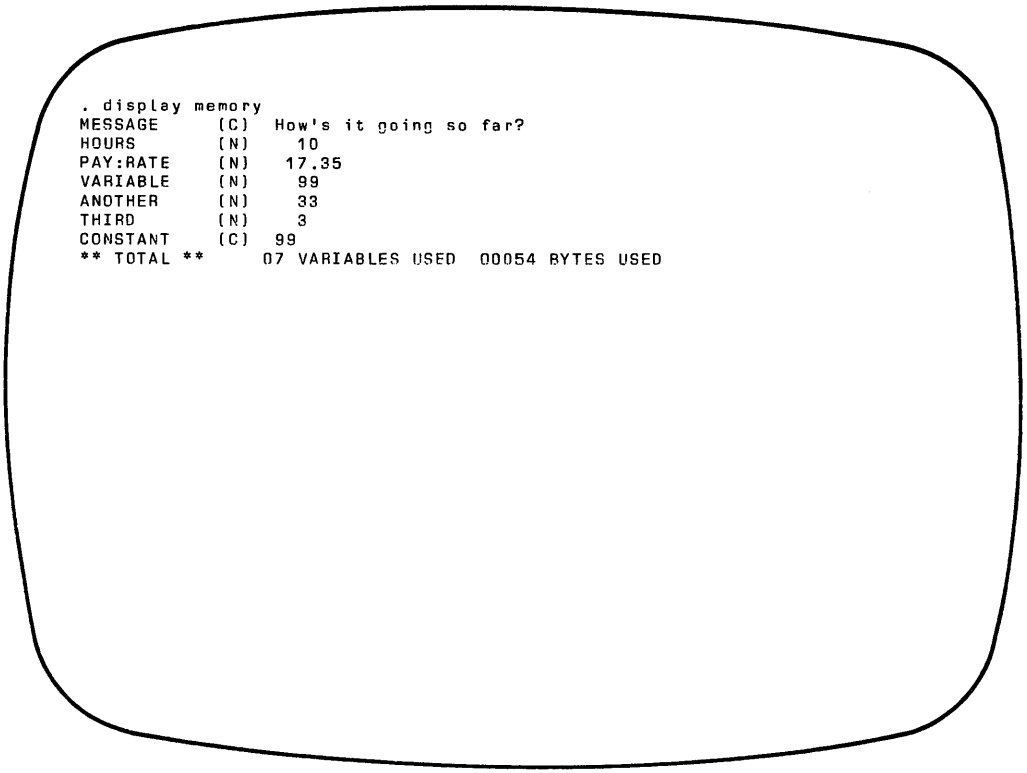
If the ampersand (&) is included as a character, it must be between two spaces. This insures that dBASE II does not interpret the character as indicating the marco substitution function described later in this chapter.

Creating Numeric Memory Variables

Numeric memory variables can be created using the STORE command. They can also be created automatically by the COUNT and SUM commands (see Chapter 10).

Displaying Memory Variables

DISPLAY MEMORY and LIST MEMORY are special forms of these commands that show the contents of the memory variable area. Try entering one of those commands. Your screen display will look like the one that follows.



```
. display memory
MESSAGE     (C)  How's it going so far?
HOURS       (N)    10
PAY:RATE    (N)   17.35
VARIABLE    (N)    99
ANOTHER     (N)    33
THIRD       (N)    3
CONSTANT    (C)  99
** TOTAL **      07 VARIABLES USED  00054 BYTES USED
```

The command lists each memory variable name, data type, and value. It also presents a count of the number of variables and the total number of characters (*bytes*) they use.

Eliminating Memory Variables

You can delete all or selected memory variables with the RELEASE command. This frees the space that the variables occupied and makes the space available for new memory variables. RELEASE has two forms:

RELEASE <memvar list>
RELEASE ALL

Enter these commands at the dot prompt.

display memory
release another
list memory
release all
display memory

```
. display memory
MESSAGE     (C)  How's it going so far?
HOURS       (N)    10
PAY:RATE    (N)   17.35
VARIABLE    (N)    99
ANOTHER     (N)    33
THIRD       (N)    3
CONSTANT    (C)  99
** TOTAL **      07 VARIABLES USED  00054 BYTES USED
. release another
. list memory
MESSAGE     (C)  How's it going so far?
HOURS       (N)    10
PAY:RATE    (N)   17.35
VARIABLE    (N)    99
THIRD       (N)    3
CONSTANT    (C)  99
** TOTAL **      06 VARIABLES USED  00048 BYTES USED
. release all
. display memory
** TOTAL **      00 VARIABLES USED  00000 BYTES USED
```

The above screen shows the different effects of the two forms of RELEASE.

Saving Memory Variables

Memory variables can be created and used for a single dBASE II session or saved in memory files and reused from session to session. The SAVE command writes all current memory variables into a memory file.

SAVE TO <mem file>

The memory file name can be any valid dBASE II file name. The system automatically appends the extension ".MEM" to memory file names.

Reading Memory Files

RESTORE reads a file of memory variables created with the SAVE command. The memory variable values are read into the system's memory variable space. All memory variables which were defined prior to the RESTORE command are deleted.

RESTORE FROM <mem file>

The following series of commands demonstrates how memory files work. Try them.

```
release all
store 1 to one
store 'abcdefghijkl' to alfabet
store alfabet +' new stuff' to chars
display memory
save to memfile
list files like *.mem
release all
display memory
store 3 to three
display memory
restore from memfile
display memory
```

```
. release all
. store 1 to one
  1
. store 'abcdefghijkl' to alfabet
abcdefghijkl
. store alfabet + '  new stuff' to chars
abcdefghijkl  new stuff
. display memory
ONE             (N)    1
ALFABET         (C)  abcdefghijkl
CHARS           (C)  abcdefghijkl  new stuff
** TOTAL **        03 VARIABLES USED  00041 BYTES USED
. save to memfile
. list files like *.mem

MEMFILE.MEM

. release all
. display memory
** TOTAL **        00 VARIABLES USED  00000 BYTES USED
. store 3 to three
  3
. display memory
THREE           (N)    3
** TOTAL **        01 VARIABLES USED  00006 BYTES USED
. restore from memfile
. display memory
ONE             (N)    1
ALFABET         (C)  abcdefghijkl
CHARS           (C)  abcdefghijkl  new stuff
** TOTAL **        03 VARIABLES USED  00041 BYTES USED
```

NOTE

Once created or read into memory, memory variable values remain intact until they are overlaid by other variables (using RESTORE) or the dBASE II session is ended. This means the variables are available to you interactively and to command files used during the session.

MACRO SUBSTITUTION FUNCTION

Memory variables can be used with the *macro substitution function* to save typing. (A function is a special purpose command, like the "!" you used in Chapter 3 to convert lowercase letters to uppercase.) When dBASE II finds an ampersand (&) followed by the name of a character string memory variable in a command, it executes the macro substitution function. The system replaces the & and memory variable name with the memory variable's contents. So, you can create memory variables consisting of frequently used character strings such as long commands or

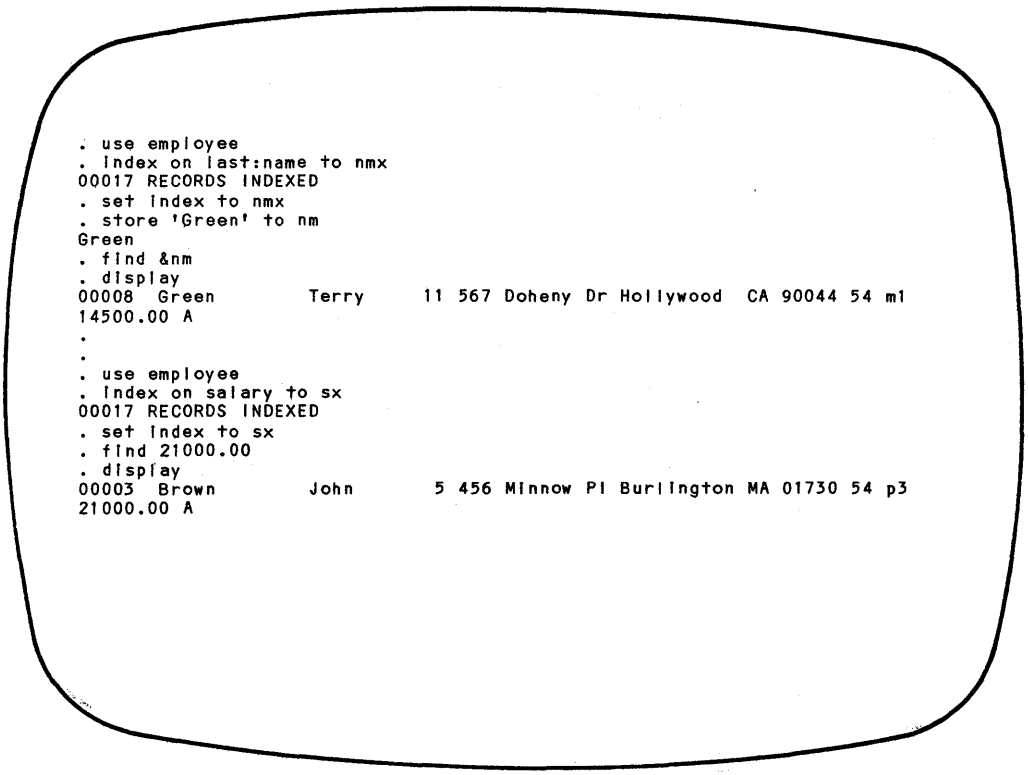
complex expressions. Assign them short names, then use them with the macro substitution function to generate long commands with just a few keystrokes.

```
. store 'salary > 20000 .and. dept:num = "89"' to c
salary > 20000 .and. dept:num = "89"
. use employee
. list for &c
00002  Berger       Mary       6 10 Kearney Rd Needham    MA 02194 89 p8
30000.00 A
00005  Embry        Albert     2 345 Sage Ave  Palo Alto  CA 94303 89 p8
22200.00 A
00007  Gilbert      Diane     14 280 Cactus Wy Las Cruces NM 88001 89 p3
24500.00 A
```

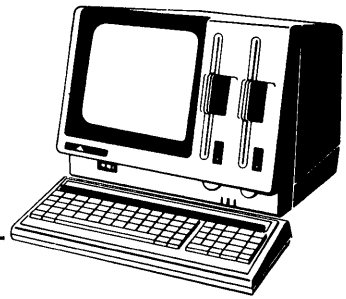
If the & is not followed by a valid memory variable name, no expansion is attempted.

USING MEMORY VARIABLES WITH THE FIND COMMAND

Memory variables can be used as objects of FIND commands. In this case, the memory variable must be used with the macro substitution function. The first command sequence in the following screen demonstrates how a name stored in a memory variable can be the object of a FIND.



```
. use employee
. index on last:name to nmx
00017 RECORDS INDEXED
. set index to nmx
. store 'Green' to nm
Green
. find &nm
. display
00008  Green          Terry      11 567 Doheny Dr Hollywood  CA 90044 54 m1
14500.00 A
.
.
. use employee
. index on salary to sx
00017 RECORDS INDEXED
. set index to sx
. find 21000.00
. display
00003  Brown          John        5 456 Minnow Pl Burlington MA 01730 54 p3
21000.00 A
```



Chapter 13

Using More Than One Data Base File

This guide uses two files for demonstration: EMPLOYEE.DBF and JOBDET.DBF. Together, the files form the personnel data base. The data base was set up as two files for ease of maintenance, non-redundancy of data, and good use of storage space. To make the best use of the files, however, you must be able to integrate them. That way, you can answer questions like "What is employee Green's job title?" and "How many employees in each department earn salaries above the midpoint for their salary range?". To answer these questions, you need information from more than one file.

USING TWO DATA BASES CONCURRENTLY

Up to this point, you have used only one data base file at a time. dBASE II has the capability of working with two files concurrently and independently. You can compare information between the files, use the information from one to update the other, and combine the files to form a new file.

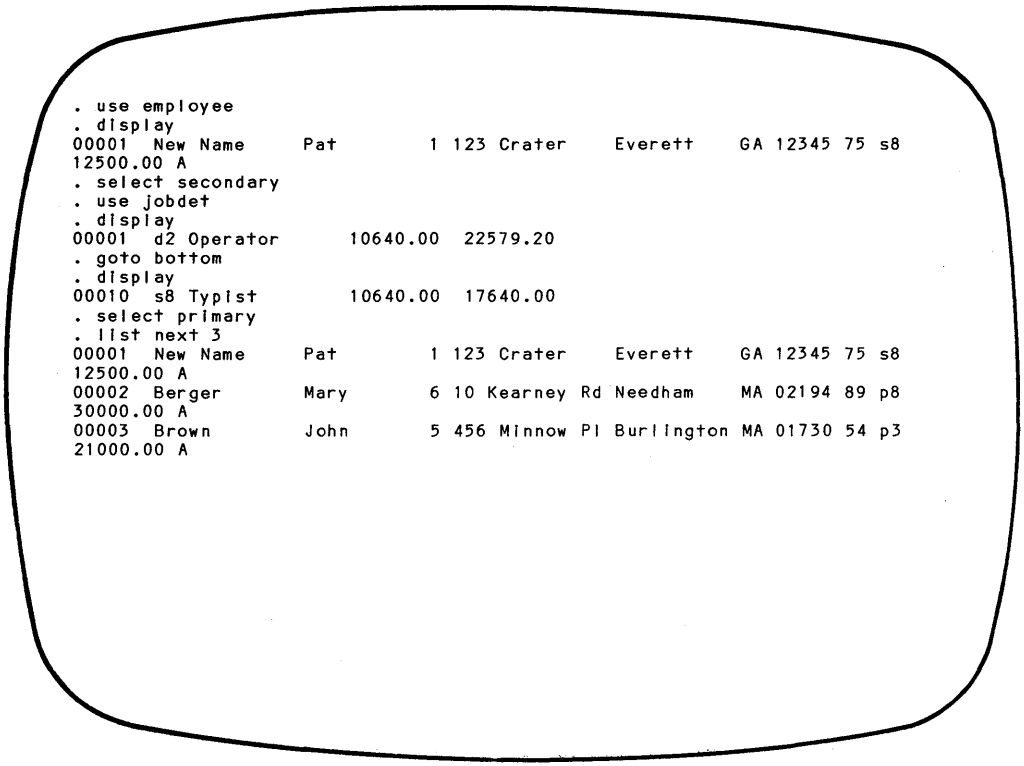
When two files are used at the same time, the system processes them in two different areas called the primary and secondary areas. dBASE II automatically selects the primary area when you USE a file, unless you tell it otherwise. Recall that when you DISPLAY STRUCTURE, the data base file in USE is called the "PRIMARY USE DATA BASE". To activate the secondary area, use the SELECT command and name the secondary area. Use the following procedure to set up two file areas.

1. USE one file. The system automatically assigns the file to the primary area.
2. Issue the following command:
   select secondary
3. USE the second file. It is assigned to the secondary area.

Access the two files by using the SELECT command to switch between the active areas. The system maintains a position in each file. The complete form of the SELECT command is:

```
SELECT [PRIMARY]
       [SECONDARY]
```

The two data base files can be processed concurrently. dBASE II commands operate on the currently selected data base file. To change the SELECTed file, issue the SELECT command.



```
. use employee
. display
00001  New Name      Pat         1 123 Crater     Everett     GA 12345 75 s8
12500.00 A
. select secondary
. use jobdet
. display
00001  d2 Operator       10640.00   22579.20
. goto bottom
. display
00010  s8 Typist         10640.00   17640.00
. select primary
. list next 3
00001  New Name      Pat         1 123 Crater     Everett     GA 12345 75 s8
12500.00 A
00002  Berger        Mary        6 10 Kearney Rd  Needham     MA 02194 89 p8
30000.00 A
00003  Brown         John        5 456 Minnow Pl  Burlington  MA 01730 54 p3
21000.00 A
```

dBASE II commands that change the position of the current record pointer affect only the currently selected data base file. These commands include GOTO, SKIP, REPORT, SORT, COPY, LIST, DISPLAY (for scope of more than one), and others.

NOTE

There is no effect if SELECT SECONDARY is entered when the secondary area is already selected or if SELECT PRIMARY is entered when that area is already active.

TRANSFERRING INFORMATION BETWEEN AREAS

To answer the question "What is employee Green's job title?" the system must access the EMPLOYEE file for last name and job code and the JOBDET for job title. Variables from both files are needed in one command. The screen that follows demonstrates how to do this.

```
. use employee
. locate for last:name = 'Green'
RECORD: 00008
. select secondary
. use jobdet
. list last:name job:title for p.job:code=s.job:code .and. p.last:name='Green'
00003  Green        Team Leader
```

Notice that the DISPLAY command uses variables from both the primary and secondary areas. When variables in the two open files have the same name, those from the area that is not in USE must be prefixed with "P." (Primary) or "S."

(Secondary) to denote the source. It is good practice to prefix all variables in the non-selected area.

```
. use jobdet
. select secondary
. use employee
. select primary
. join to newfile for p.job:code=s.job:code field last:name,job:code,job:title
. use newfile
. list
00001  Harris       d8 Shift Leader
00002  Shaffer      d8 Shift Leader
00003  Green        m1 Team Leader
00004  Brown        p3 Programmer
00005  Destry       p3 Programmer
00006  Gilbert      p3 Programmer
00007  Rowland      p3 Programmer
00008  Smyth        p4 Prog/Analyst
00009  Berger       p8 Sr Programmr
00010  Embry        p8 Sr Programmr
00011  Freitag      p9 Sr PrgAnlyst
00012  Inders       p9 Sr PrgAnlyst
00013  New Name     s8 Typist
00014  Green        s8 Typist
00015  Howser       s8 Typist
00016  Peters       s8 Typist
00017  Schaller     s8 Typist
```

When using two areas, the system can compare and select data from both areas, but can only reposition the current record pointer in one area.

In the previous example, JOBDET.DBF is selected as the primary use data base and EMPLOYEE.DBF is the secondary use data base. When the files are first USEd, the system is positioned on the first record in each file. The current record pointer is moved to the last record in the primary area. Then, the secondary area is selected. When the FOR phrase is evaluated in the LIST command, each EMPLOYEE record is compared, in turn, with the last JOBDET record. The current record pointer moves sequentially through the EMPLOYEE file, while the system maintains its position in the JOBDET file. This process is shown in Figure 13-1.

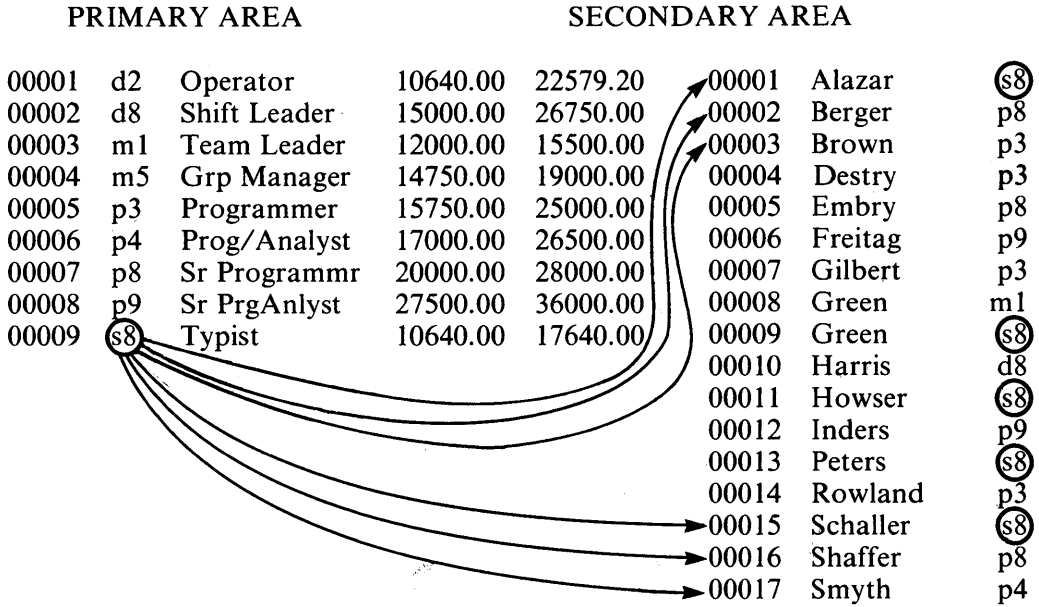


Figure 13-1 Current Record Pointer Movement with Two Files in Use

When using sequential commands, it is important to SELECT the area containing the file to be processed sequentially before issuing any file processing commands.

SET LINKAGE ON can be used to make sequential commands perform positioning on both the primary and secondary data base files. The effect is that of a single data base with up to 64 fields and 2000 characters per record. Use prefixes to distinguish field names that are the same in both data files.

COMBINING TO FORM A NEW FILE

JOIN creates a new file from elements of existing files, two files at a time. The files must be related to each other in some way. The complete form of the JOIN command is:

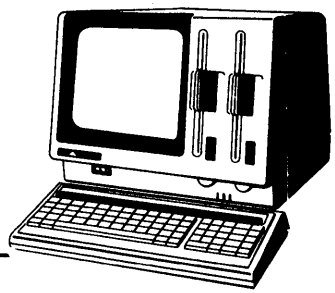
JOIN TO <file> FOR <exp> [FIELD <field list>]

- The FOR phrase sets the criteria for joining the files. It can be a simple expression (PART:NO = S.PART:NO) or a complex logical expression.
- The FIELD phrase is optional. If it is omitted, the new file consists of all fields in the PRIMARY USE file's structure plus as many of the SECONDARY USE file's fields as will fit before exceeding dBASE II's limit of 32 fields per file.
- If the FIELD phrase is included, the new file consists of the named fields only. The fields may be selected from both files. Be sure to use commas in the field list to separate the field names. Do not prefix the field names unless the same name is used in both files.
- Before issuing the JOIN command, SELECT PRIMARY and position the current record pointer on the first record of that file.
- If the new file does not exist, it is created with this command. If the file already exists, it is completely replaced with the results of the JOIN command.

```
. use jobdet
. select secondary
. use employee
. select primary
. join to newfile for p.job:code=s.job:code field last:name,job:code,job:title
. use newfile
. list
00001  Harris       d8 Shift Leader
00002  Shaffer      d8 Shift Leader
00003  Green        m1 Team Leader
00004  Brown,       p3 Programmer
00005  Destry       p3 Programmer
00006  Gilbert      p3 Programmer
00007  Rowland      p3 Programmer
00008  Smyth        p4 Prog/Analyst
00009  Berger       p8 Sr Programmr
00010  Embry        p8 Sr Programmr
00011  Freitag      p9 Sr PrgAnlyst
00012  Inders       p9 Sr PrgAnlyst
00013  New Name     s8 Typist
00014  Green        s8 Typist
00015  Howser       s8 Typist
00016  Peters       s8 Typist
```

In operation, JOIN positions dBASE II on the first record of the primary file and evaluates the FOR expression for each record in the secondary file. Each time the expression yields a TRUE result, a record is added to the new data base. When the end of the secondary file is reached, the primary file is advanced one record, the secondary file is repositioned on the first record, and the FOR expression is evaluated for each record in the secondary file. Then, the comparison is made for the third record of the primary file, and so on. This process continues until the primary file is exhausted.

The JOIN command takes a lot of time to complete if the contributing data bases are large. If the JOINing criteria are too loose, causing many joinings per primary record, there is the potential for causing a JOIN that dBASE II cannot complete. For example, suppose that the primary and secondary USE files each contain 1000 records, and that the FOR expression is always true. A million records should be output by the JOIN into a data base whose size would exceed the dBASE II maximum of 65,535 records.



Chapter 14

Command Files

Up to this point, every time you needed to execute a command you entered it. Once you are familiar with dBASE II, you will find yourself repeating sequences of commands again and again. You may use one routine to update files, another to produce a report, a third to display the answer to a frequently asked question. dBASE II can save instruction sequences in *command files* that you call and execute with just one statement.

A *command file* consists of dBASE II commands. When you execute a command file, the system reads the command list from diskette and starts at the first line, processing the commands one at a time until the list is finished.

SETTING UP COMMAND FILES

Command files can be created and modified by text editors and word processors that produce CP/M files, and with the MODIFY COMMAND capability in dBASE II. Regardless of how they are created, command files must conform to the naming conventions and file structures the dBASE II system uses. Consult your word processor or text editor manual for instructions to use them or use the following dBASE II command to write command files.

MODIFY COMMAND [<command file>]

If the command file name is omitted, the system prompts you for the file name. You may enter the name with or without the extension ".PRG" (for PROGRAM).If the file does not exist, it is created with ".PRG" as the file type extension. If the file does exist, changes are made to the copy of the file with the ".PRG" extension. A backup of the original unchanged file with the extension ".BAK" is automatically created.

Editing Command Files in dBase II

The MODIFY COMMAND instruction creates a command file if it does not already exist, or retrieves an existing file. dBASE II erases the screen, displays the file contents, and puts the system in full-screen edit mode. Table 14-1 lists the dBASE II commands to use when editing command files.

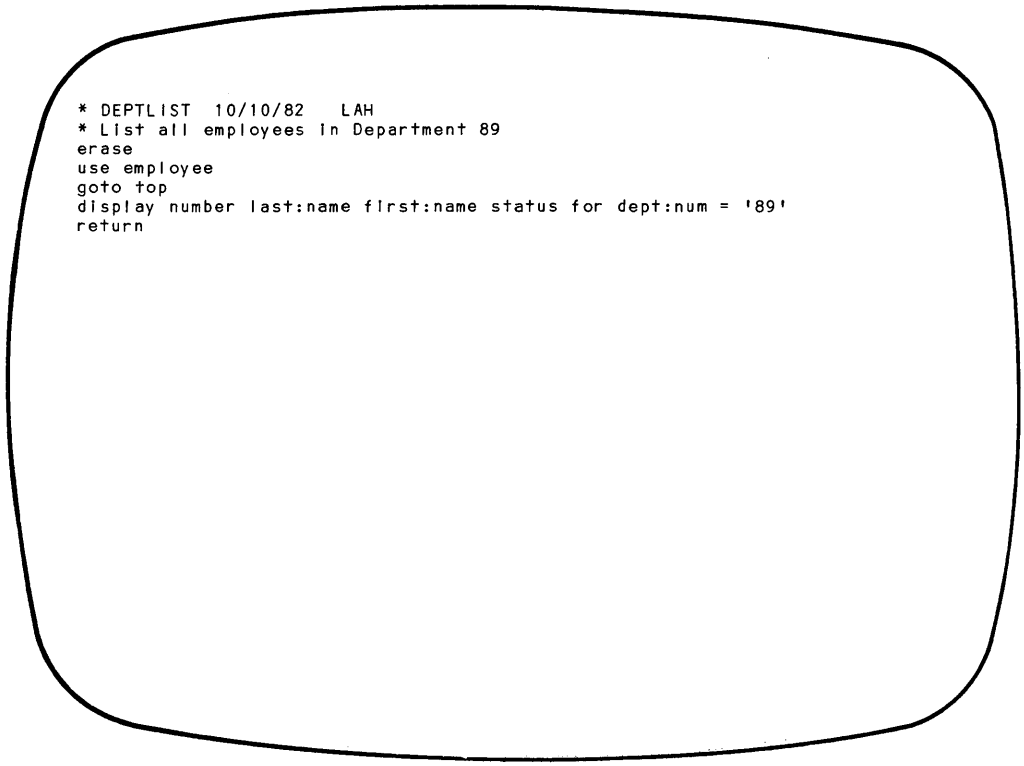
Table 14-1 Full Screen Edit Commands - MODIFY COMMAND

COMMAND	ACTION
→ **CTRL D**	Moves cursor ahead one character
← **CTRL S**	Moves cursor back one character
DEL **CTRL G** **CTRL X**	Deletes character under the cursor
CTRL C	Advances one screen
CTRL R	Moves down one line
CTRL W	Saves changes and returns to dBASE II command mode
CTRL Q	Aborts changes and returns to dBASE II command mode
INS **CTRL V**	Toggles insert/overtype modes
PRINT **CTRL P**	Toggles printer on/off
CTRL T	Deletes line under cursor
CTRL N	Inserts blank line under cursor

To see how MODIFY COMMAND works, create a file called DEPTLIST.PRG that produces the "SALARY EXPENSE BY DEPARTMENT" report defined in Chapter 10. Enter the following command at the prompt:

```
modify command deptlist
```

Since this is a new file, when the screen is erased the only thing you see is the cursor in the upper left corner. Enter the command file displayed in the following screen.



```
* DEPTLIST  10/10/82   LAH
* List all employees in Department 89
erase
use employee
goto top
display number last:name first:name status for dept:num = '89'
return
```

Use the cursor control commands in Table 14-1 to make corrections and changes. When the file is completely entered, press **CTRL W** to save the file and return to command mode.

NOTE

The editor can back up only about 5,000 lines, so plan to edit moving forward through the text on larger files.

Using Other Systems to Create Command Files

Command files created using text editors and word processors must conform to the command file structure that dBASE II generates automatically.

- Each physical command line can be a maximum of 77 characters, including the semicolon signaling that the line is continued or the carriage return and line feed characters that signal end-of-record.
- Logical command lines can be a maximum of 256 characters, including semicolons and the carriage return and line feed characters.
- Tab characters can be entered, but they are converted to single spaces.

COMMAND FILE CONTENTS

Command files can contain any dBASE II commands. However, be careful in selecting the commands that are included. If you do not want the person using the command file to enter data through the keyboard, do not use commands that require user input (like INSERT, CREATE, APPEND, REPORT, and so on).

The sample file (DEPTLIST.PRG) produces the "SALARY EXPENSE BY DEPARTMENT" report. It uses commands you already know plus several new ones.

Documenting the Procedures

It is a good idea to include comments in command files explaining the processing. Programmers call this *documentation.* The comments are not printed or displayed when the command file is executed. They are simply in the file for your own information. dBASE II has two commands that allow the insertion of text in command files.

```
* <char string>
NOTE <char string>
```

The commands are identical in the rules for their formation and the results they generate. There must be at least one space between the command and the characters that form the comment. The comment command cannot be on the same line with another command.

Communicating with the System User

Comments and instructions can also be created for display and printing with the REMARK command. REMARK stores comments in the command file and (unlike NOTE) also displays and prints them as comments or prompts when the file is executed.

REMARK <char string>

There must be at least one space between the command and the characters that form the remark. The REMARK command cannot be on the same line with another command.

Housekeeping

ERASE clears the screen. It is a one-word command that does not take any modifiers.

Ending a Command File

A command file is closed when the end of the file is reached or when the RETURN command is issued. (The RETURN command is the world "RETURN", not the key labelled **RETURN** on the APC keyboard.) This command is usually the last executable instruction in a command file. RETURN is not always necessary, but it is a good practice to include it.

EXECUTING COMMAND FILES

Command files are started by the DO command, which names the file containing instructions for the system to execute.

DO <command file>

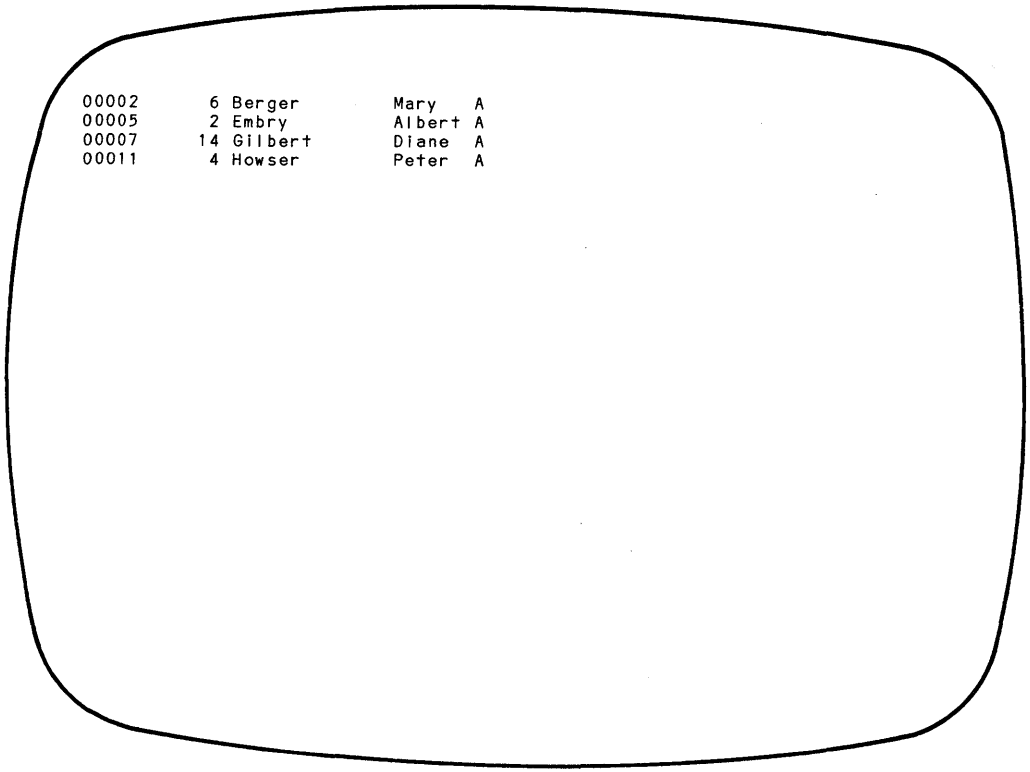
- The named file must be stored with either the ".PRG" or ".BAK" extension. If the extension is not included in the name, ".PRG" is assumed.
- <command file> must consist entirely of commands.

When the DO command is executed, the system opens the command file and reads it. The commands are interpreted and executed just as keyboard commands are. (The DO command is fully explained in Chapter 17.)

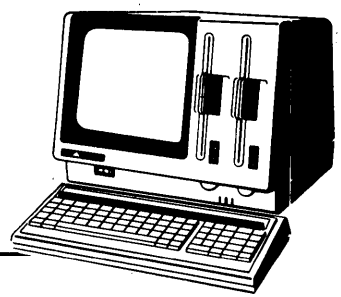
Test the command file you just created by entering the following DO command:

do deptlist

If you made a mistake when you typed in the file, your display will not look like the one that follows. Instead, the system will display a standard error message. Enter **MODIFY COMMAND DEPTLIST** and compare your file to the sample. Edit the command file as necessary and try to DO command again.



When you create a command file, you are actually writing a dBASE II program. Chapters 17 through 19 describe the dBASE II programming capabilities and commands in detail.



Chapter 15

Functions

Functions are special purpose operations that accomplish things that are difficult or impossible to do using regular arithmetic, logical, and string operations. In dBASE II, there are three types of functions: numeric, character, and logical. The function type is determined by the type of value that the function generates. Table 15-1 lists the dBASE II functions by type.

In general, functions are called by a name or symbol followed by an expression enclosed within parentheses. Be sure to use the parentheses when entering functions. Do not worry about memorizing the functions, but do scan the descriptions in this chapter so that you know the kinds of things functions can do and where to look for information about them.

Table 15-1 Functions

FUNCTION NAME	TYPE
Data Type	Character
Date	Character
Deleted Record	Logical
End-of-File	Logical
File	Logical
Integer	Numeric
Macro Substitution	Character
Number to Character	Character
Peek	Numeric
Record Number	Numeric
String	Character
String Length	Numeric
String to Numeric	Numeric
Substring	Character
Substring Search	Numeric
Test	Numeric
Trim	Character
Uppercase	Character

NUMERIC FUNCTIONS

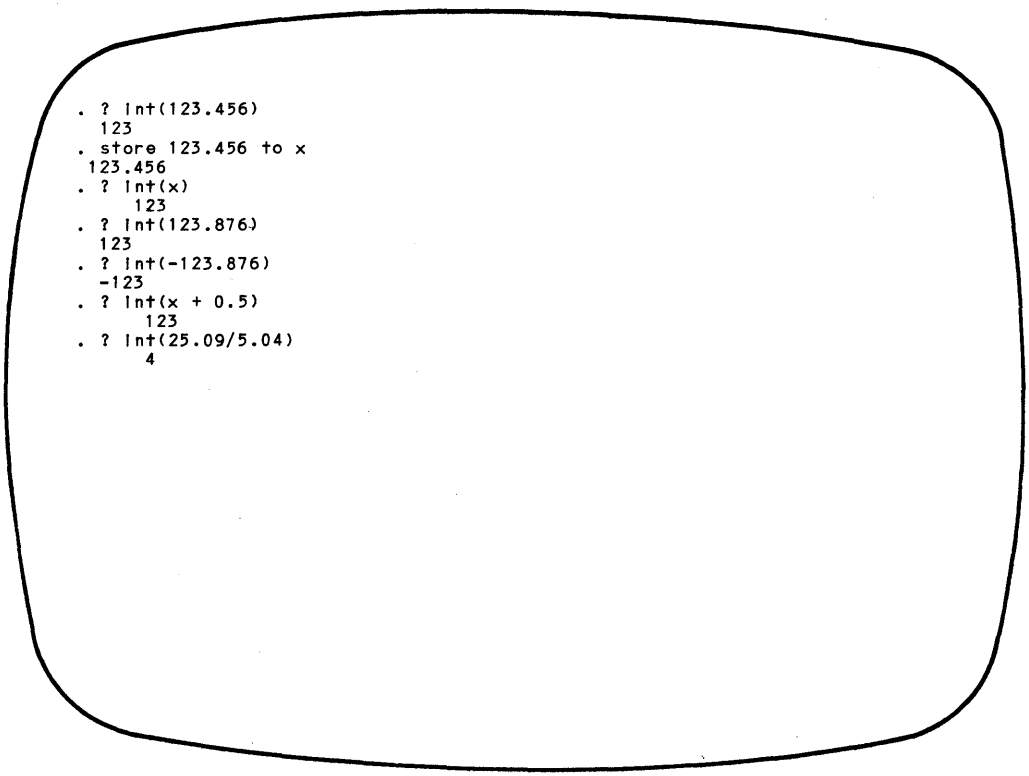
Integer Function

INT(<number/var/exp>)

The integer function evaluates a numeric expression and discards the fractional part, if any, to yield an integer value. The value returned is the *truncated* - not rounded - value of the numeric expression within parentheses. The term inside the parentheses can be a number, numeric variable, or simple or complex expression. Expressions are evaluated and then the result is truncated.

The following examples demonstrate some properties of the integer function and ways it can be used. Enter these commands at the dot prompt, or try some examples of your own.

```
? int(123.456)
store 123.456 to x
? int(x)
? int(123.876)
? int(-123.876)
? int(x + 0.5)
? int(25.09/5.04)
```



Notice that both positive and negative numbers can be used. A call to a variable returns the truncated integer formed from the current value of that variable. The result of the function can be stored in a file or memory variable.

To round to the nearest whole number, add .5 to the value of the expression. The expression within parentheses is first evaluated, and then the integer function of the result is taken.

The integer function can be used to round a value to any number of decimal places. INT(<value> * 10 + .05)/10 rounds a value to the nearest tenth because of the order of precedence of operations - parentheses, then integer, then division. To round to two places, use "100" in place of the "10"s. For three places, use "1000", and so on.

Peek Function

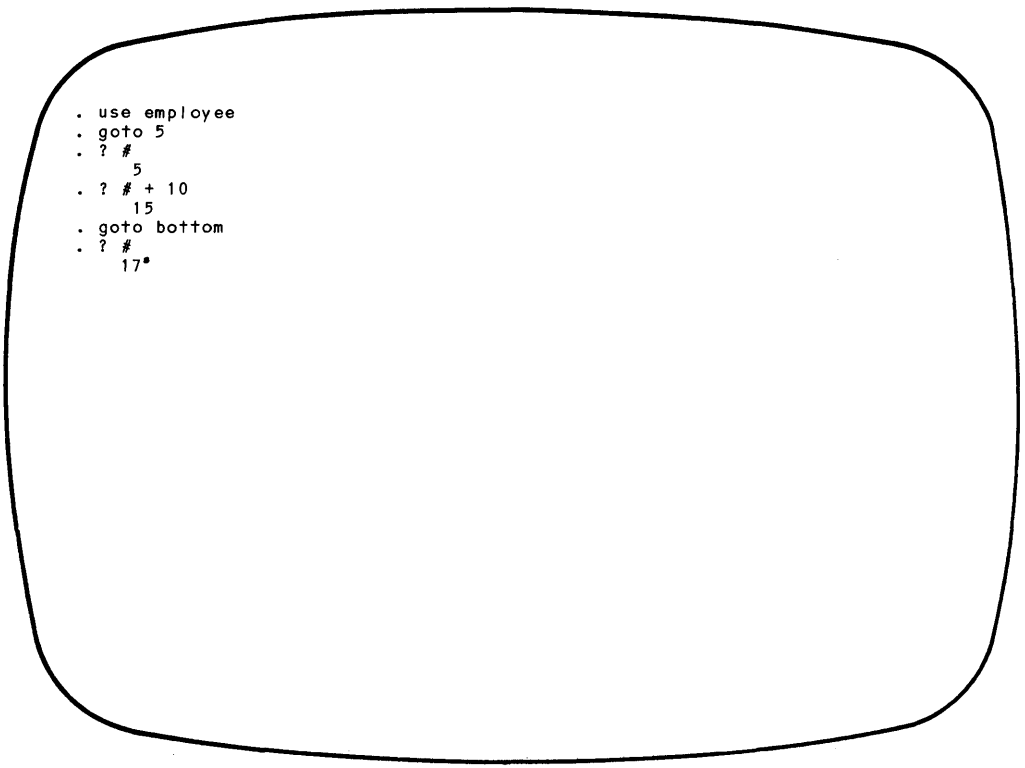
PEEK (<address>)

The peek function returns the decimal value of the byte at the specified address in the computer's memory. The address must be in decimal format. This function is used by advanced programming operations.

Record Number Function

#

The record number function, #, returns the integer corresponding to the current record number. The result is a number that can be stored, displayed, and used in operations.

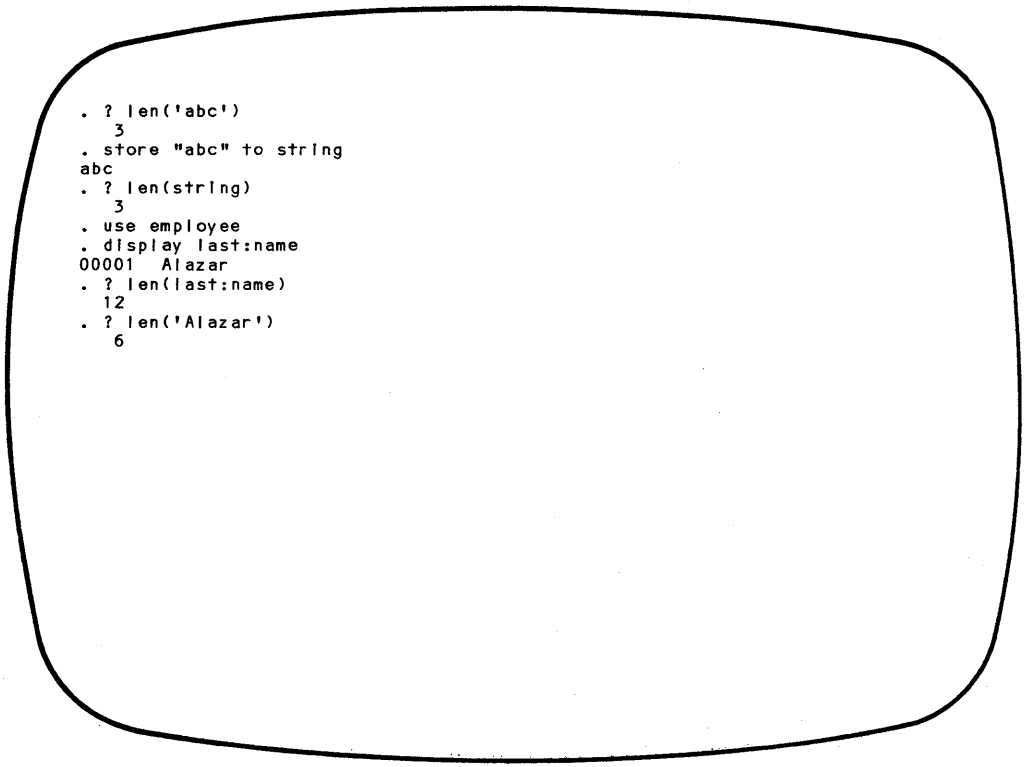


String Length Function

LEN(<var/char string>)

The string length function returns the number of characters in the named variable or string. If a character type variable is used, this function returns the size of the field, not the length of the contents. To test the difference, enter the following commands.

```
? len('abc')
store 'abc' to string
? len(string)
use employee
display last: name
? len(last:name)
```



```
. ? len('abc')
   3
. store "abc" to string
abc
. ? len(string)
   3
. use employee
. display last:name
00001   Alazar
. ? len(last:name)
  12
. ? len('Alazar')
   6
```

The string length function can be useful when a program has to determine, without operator intervention, how much storage to allocate for information.

String to Numeric Function

VAL(<char string>)

The string to numeric function converts a character string or substring consisting of numerals and a sign, if present, into the equivalent numeric quantity.
The length of the integer returned is equal to the number of characters in the string. If the character string begins with numeric characters but contains non-numeric characters, the value generated by the function is the value of the leading numeric characters only. The system interprets a period as a non-numeric character, not a decimal point. Therefore, the function truncates the numerals to a whole number value.

The following screen demonstrates how the function works.

```
. ? val('123')
 123
. ? val('122xxx')
    122
. ? val('123.456')
     123
. store '123.456' to num
123.456
. ? 14 + &num
  137.456
```

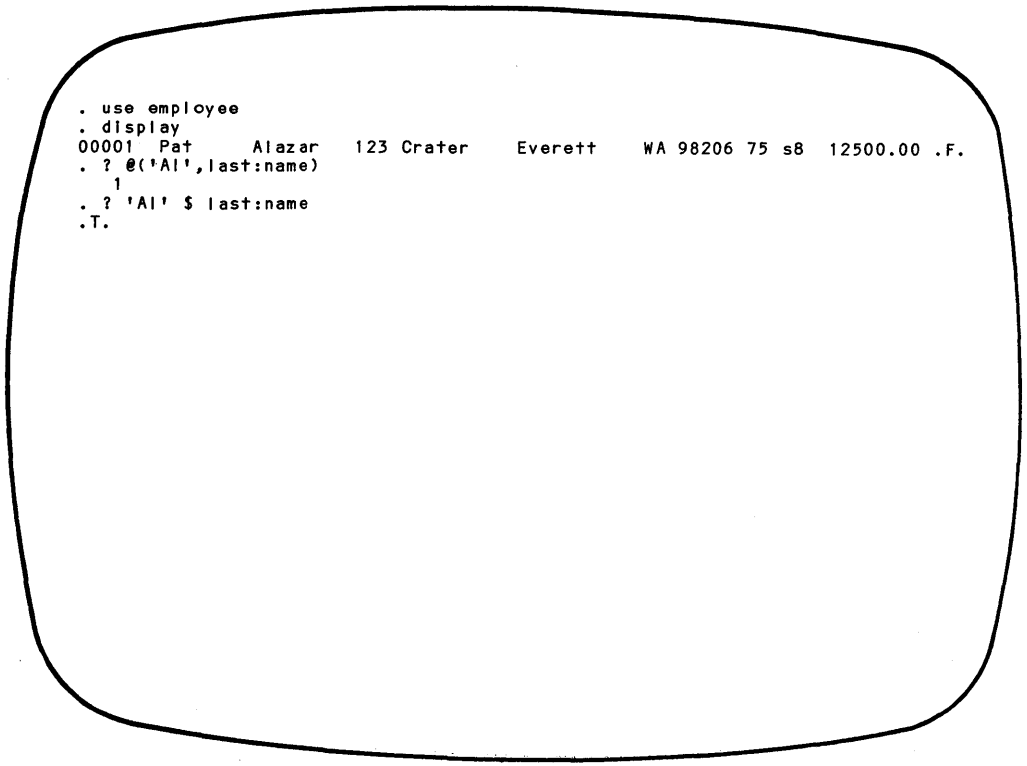
Another way to convert character strings of numerals into numeric values is the macro substitution function (&). This function is demonstrated in the previous screen and is described in Chapter 12 and later in this chapter. The macro substitution function will convert a character string, including the decimal point and sign if present, into a numeric value.

Substring Search Function

@(<char string1>,<char string2>)

Think of the substring search function as a way of asking the system "Where is string1 in string2?" The function returns an integer whose value is the character position which begins a substring identical to <char string1>. If the first string does not occur in the second string, a value of 0 is returned.

One use for this function is to determine where a specific string starts so that you can use that number for <start> in the substring function (described later in this chapter). The function can also tell whether a specific string occurs at all. Command files can use this information when searching for strings without operator intervention.



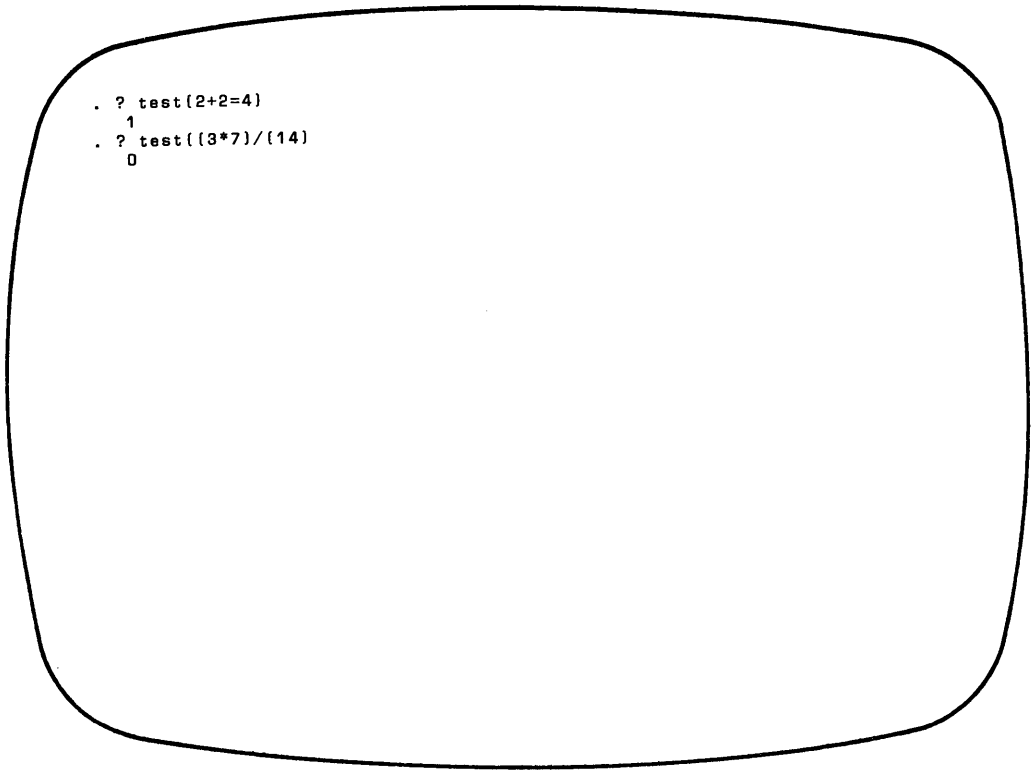
```
. use employee
. display
00001  Pat      Alazar   123 Crater    Everett    WA 98206 75 s8  12500.00 .F.
. ? @('Al',last:name)
   1
. ? 'Al' $ last:name
.T.
```

Note that the substring search function is similar to the substring logical operator ($). The substring logical operator tells whether one string occurs within another (it returns a True or False value), while the substring search function tells the starting position of one string within another string.

Test Function

TEST (<exp>)

The test function checks an expression for syntactic correctness and returns a value of zero if the expression contains a syntax error. If the expression does not contain a syntax error, a non-zero value is returned.



The test function is useful during the development and testing of command files (see Chapters 17-19). Normally, when the system encounters a syntax error, it interrupts execution of the command file and displays an error message. Using the test function, you can test a command for accurate syntax and bypass a command or take corrective action if the command contains an error. The following IF command demonstrates this use of the TEST function.

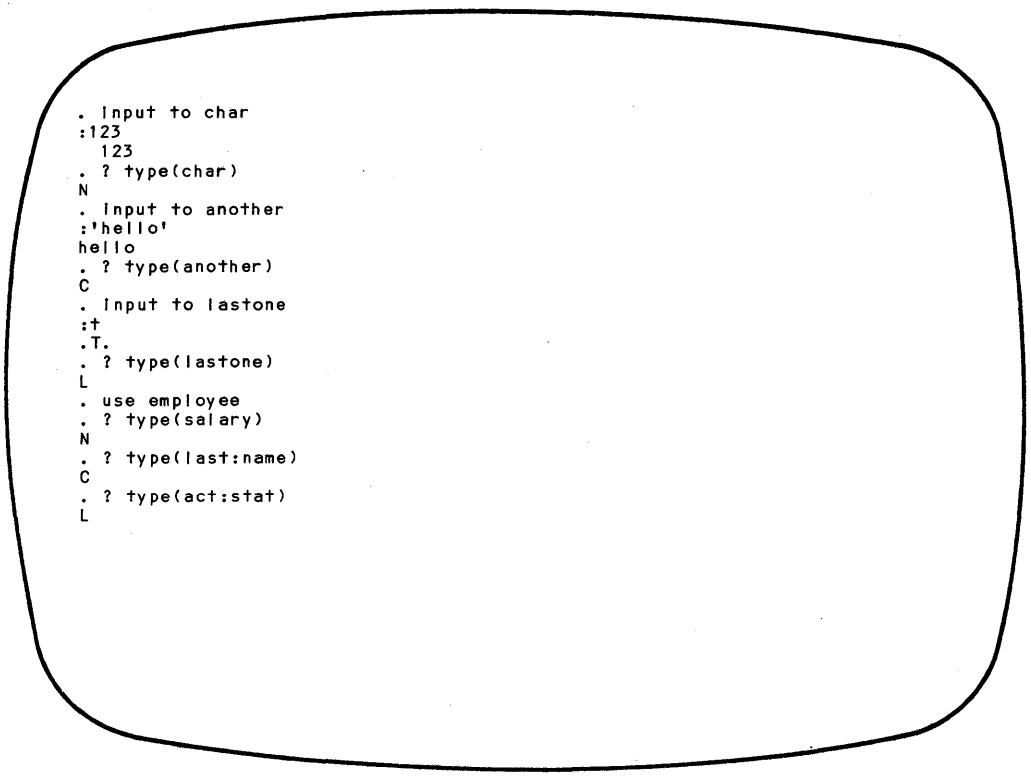
```
IF 0 = TEST (<exp>)
    take a corrective action; expression contains an error
ELSE
    proceed; expression is correct
ENDIF
```

CHARACTER FUNCTIONS

Data Type Function

TYPE(<exp>)

The data type function returns a one-character string that contains "C", "N", or "L" depending on whether the data type of the expression is Character, Numeric, or Logical respectively.



```
. Input to char
:123
  123
. ? type(char)
N
. Input to another
:'hello'
hello
. ? type(another)
C
. Input to lastone
:t
.T.
. ? type(lastone)
L
. use employee
. ? type(salary)
N
. ? type(last:name)
C
. ? type(act:stat)
L
```

Date Function

DATE()

The date function generates a character string that contains the system date in the format MM/DD/YY. The character string returned is always eight characters long. Enter the function exactly as shown, with no characters or spaces between the parentheses. The parentheses indicate to dBASE II that this is a function and not a variable named "DATE".

The date can be entered when you start a dBASE II session, and can be changed at any time using the SET DATE TO command.

```
. ? date()
00/00/00
. set date to 9 23 48
. ? date()
09/23/48
. set date to 15+68+99
. ? date()
15/68/99
```

Number to Character Function

CHR(<numeric exp>)

The number to character function returns the ASCII character equivalent of the numeric expression. For instance, if the expression is the number 13, CHR(13) generates the ASCII carriage return character. This function is useful for sending direct controls to hardware devices such as printers and for interpreting keyboard input.

Macro Substitution Function

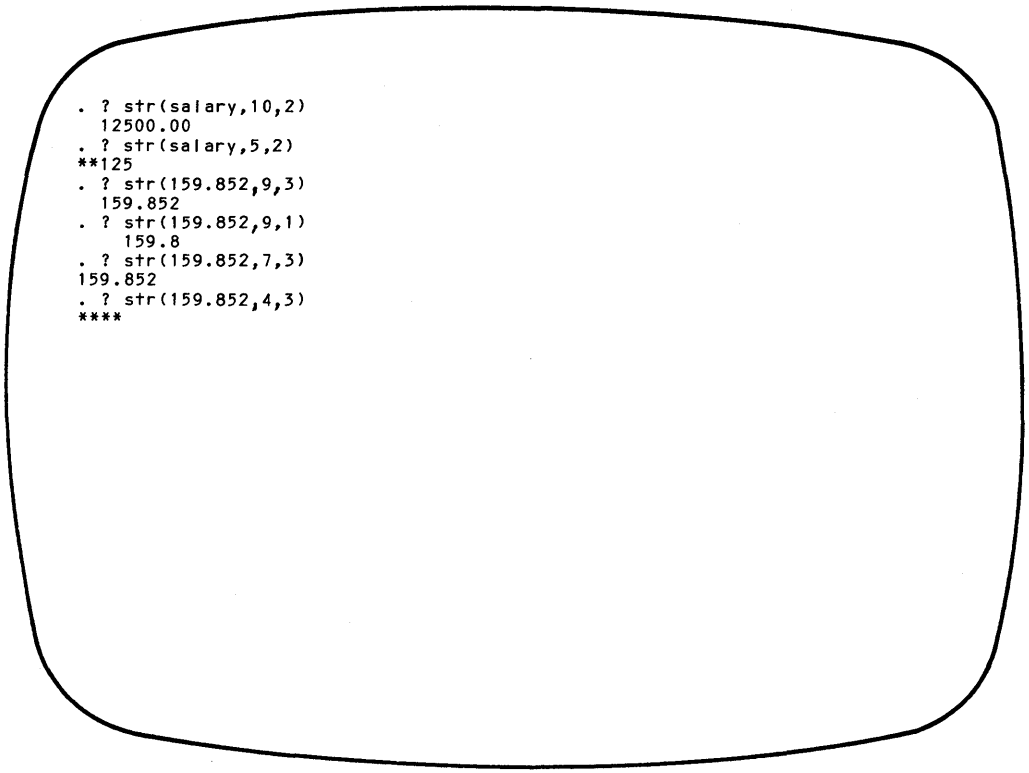
&memvar

The macro substitution function is fully described in Chapter 12. It tells the system to substitute the contents of a memory variable for the macro substitution symbol and variable name in the current command.

String Function

STR(<numeric exp/var/number>,<length>,[<dec>])

The string function converts a number or the contents of a numeric variable or expression into a string of the specified length with the specified number of digits to the right of the decimal point. The length must be at least large enough to hold all the digits plus the decimal point. If the field is not large enough to hold the result, the function returns asterisks (*) to indicate that the value is wrong. If the numeric value is shorter than the specified field, the remaining portion is filled with blanks. If the decimal portion is not specified, zero is assumed. Except when used to generate a key for indexing, all specifiers may be literals, variables, or expressions.



WARNING

When this function is used to generate a key for indexing, the specifiers must be literals. Variables and expressions are not accepted.

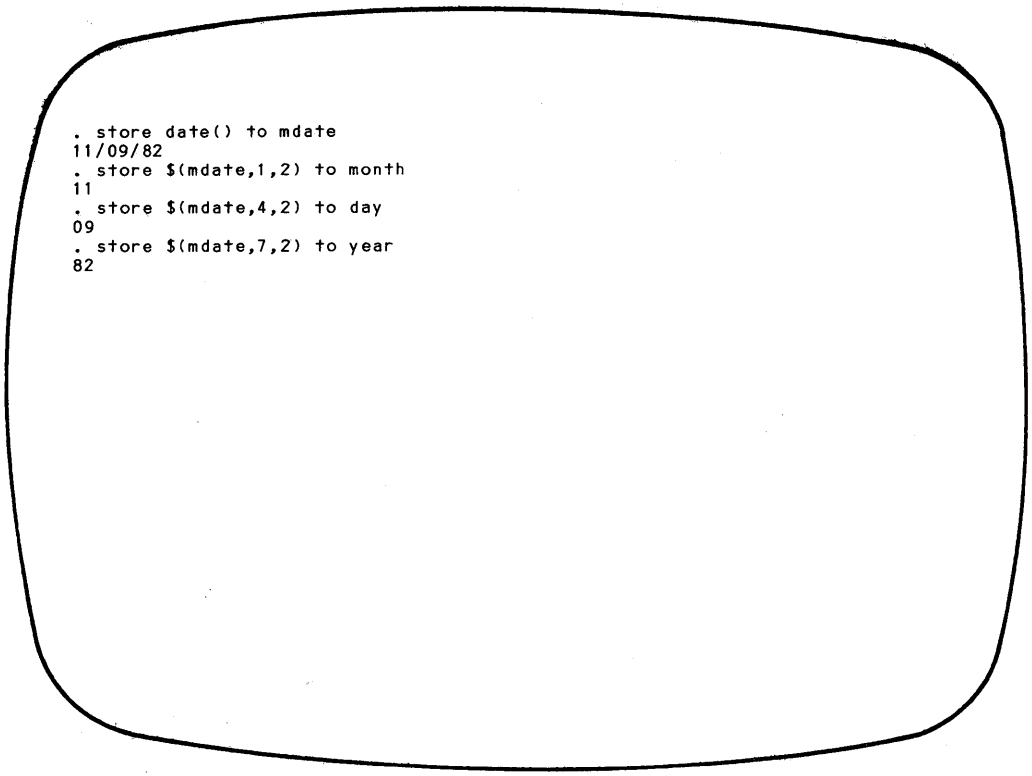
Substring Function

$(<char string>,<start>,<length>)

The substring function forms a character string from the specified part of another character string. The value of the substring is a character string of the specified length filled with characters from <char string> starting at the specified position. As with the string function, <start> and <length> may be literals, variables, or expressions unless the function is used to generate an index key.

If <length> is longer than the character string or if it is longer than the number of characters from <start> to the end of the string, the result consists of only those characters that are in the character string.

The following example demonstrates how this function can be used as the first step of a date validation routine. Groups of two characters are taken from a six character date field. Then, the results can be converted to integers (using the VAL(...) function) and checked to determine whether they are in the correct range.



```
. store date() to mdate
11/09/82
. store $(mdate,1,2) to month
11
. store $(mdate,4,2) to day
09
. store $(mdate,7,2) to year
82
```

WARNING

When the function is used to generate a key for indexing, the specifiers must be literals.

Trim Function

TRIM(<char string>)

The trim function eliminates the trailing blanks in a field. Usually dBASE II carries trailing blanks on all variables to avoid column alignment problems on displays.

A special form of the TRIM function removes the trailing blanks from a character string memory variable:

STORE TRIM(<char var>) TO <memvar>

```
. use employee
. display last:name first:name
00001  Alazar       Pat
. display trim(last:name) trim(first:name)
00001  Alazar Pat

. store 'a few characters          ' to hold
a few characters
. ? len(hold)
  26
. store trim(hold) to hold
a few characters
. ? len(hold)
  16
```

WARNING

This function must not be used as part of an index key.

Uppercase Function

!(<char string>)

The uppercase function converts all lowercase characters in the character string to uppercase. Any other characters in the string are unaffected.

This function is useful for converting keyboard input into a standard form in the files. This makes it simpler when searching for data later, since all of the data is stored in uppercase regardless of how it was entered.

```
. ? !('this will print in uppercase letters once converted')
THIS WILL PRINT IN UPPERCASE LETTERS ONCE CONVERTED
.use employee
. display last:name
00001  Alazar
. display !(last:name)
00001  ALAZAR
. display street
00001  123 Crater
. display !(street)
00001  123 CRATER
. display salary
00001    12500.00
. display !(salary)
*** SYNTAX ERROR ***
          ?
display !(SALARY)
CORRECT AND RETRY (Y/N)? N
```

LOGICAL FUNCTIONS

Deleted Record Function

*

The deleted record function returns .T. (TRUE) if the current record has been marked for deletion and .F. (FALSE) otherwise.

```
. use employee
. list next 5
00001  Pat      Alazar    123 Crater     Everett     WA 98206 75 s8  12500.00 .F.
00002  Albert   Embry     345 Sage Ave   Palo Alto   CA 94303 89 p8  22200.00 .F.
00003  Ralph    Destry    234 Mahogony   Deerfield   FL 33441 38 p3  15575.00 .F.
00004  Peter    Howser    678 Dusty Rd   Chicago     IL 60631 89 s8   9500.00 .F.
00005  John     Brown     456 Minnow Pl  Burlington  MA 01730 54 p3  21000.00 .F.
. goto top
. delete
00001 DELETION(S)
. ? *
.T.
. display next 5 for *
00001 *Pat      Alazar    123 Crater     Everett     WA 98206 75 s8  12500.00 .F. .
. goto top
. display next 5 for .not. *
00002  Albert   Embry     345 Sage Ave   Palo Alto   CA 94303 89 p8  22200.00 .F.
00003  Ralph    Destry    234 Mahogony   Deerfield   FL 33441 38 p3  15575.00 .F.
00004  Peter    Howser    678 Dusty Rd   Chicago     IL 60631 89 s8   9500.00 .F.
00005  John     Brown     456 Minnow Pl  Burlington  MA 01730 54 p3  21000.00 .F.
```

End-Of-File Function

EOF

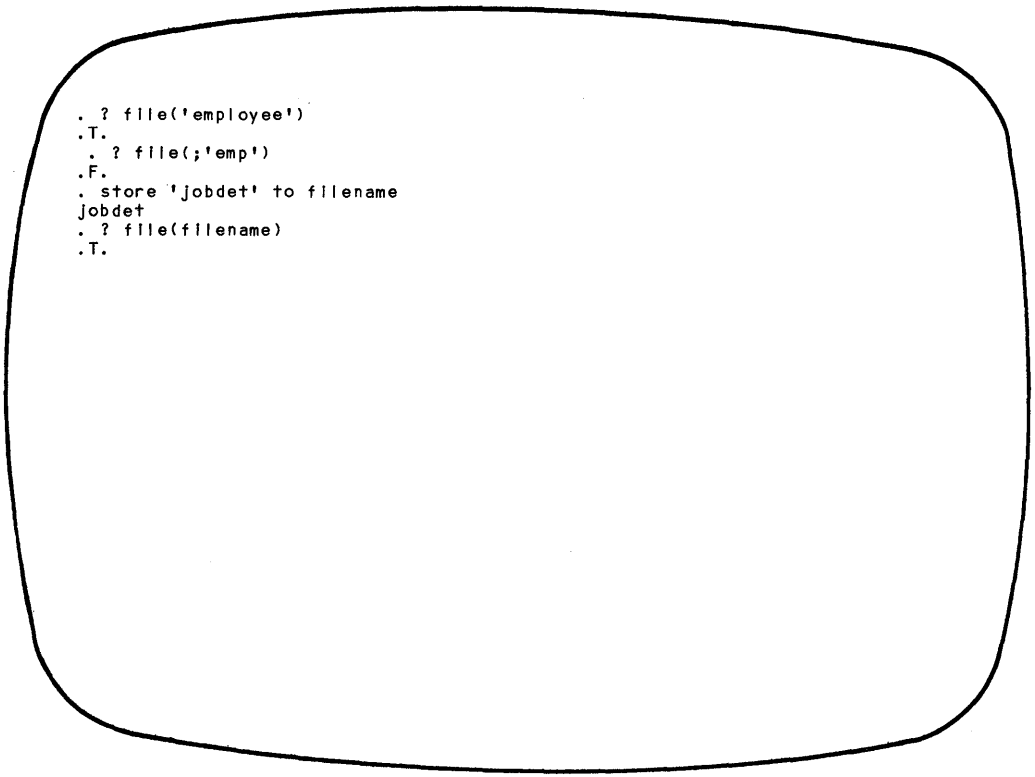
The end-of-file function returns .T. (TRUE) if the end of file has been reached for the file in USE. (This means that the current record pointer is set beyond the last record in the data base file.) Otherwise, the value is .F. (FALSE). This function is useful in command files that must perform a series of operations on each record in a file and then perform a new set of actions when file processing is complete.

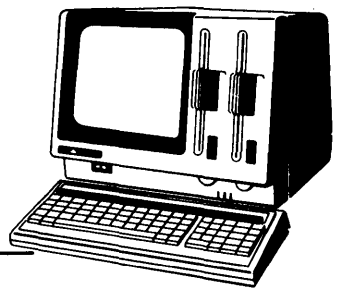
```
. use employee
. goto 5
. ? eof
.F.
. goto bottom
. ? eof
.F.
. display
00017  Per      Inders   321 Sawtelle  Tuscon      AZ 85702            0.00 .F.
. skip
RECORD: 00017
. ? eof
.T.
```

File Function

FILE(<"file name"/var/exp>)

The file function is a logical function that yields the value .T. if the named file exists and .F. if it does not. If you use a specific file name, enclose it in quote marks. The name of a string variable does not require the quote marks. You can also use any valid string expression. For instance, FILE("B:"+ database) would tell you whether the file name stored in the memory variable "database" is on Drive B. This function is useful in programs to see whether a file name entered by the user is spelled correctly.





Chapter 16

Setting Up The Environment

dBASE II has a number of commands, called SET commands, that control how the system interacts with the APC. You can change these parameters or leave them at their default values. The SET command have two formats.

The first format allows those parameters that are "toggles" to be set on or off. These commands are listed in Table 16-1. The general format of these commands is:

```
SET <parm>  [ON]
            [OFF]
```

The second format allows you to reset parameters that need string values to operate. These parameters can be changed in command files or interactively as you work at the APC. These SET commands are listed in Table 16-2. Their general form is:

```
SET <parm> TO <option>
```

There is no need to memorize the SET commands. As you work with the established defaults, you can decide whether you want to change any of the parameters. For now, just read through the list and try the commands.

Every time you initiate a dBASE II session, each parameter is set to its default value. These values are underlined in Table 16-1.

Table 16-1 SET ON and SET OFF Commands

COMMAND		ACTION
SET ALTERNATE	ON	Echo output to a disk file.
	OFF	Do not echo to a disk file.
SET BELL	ON	In data entry and edit modes, ring bell when illegal data is entered and when data completely fills a field.
	OFF	Turn bell off.
SET CARRY	ON	When inserting and appending records, carry data forward from the previous record.
	OFF	Do not carry data forward.
SET COLON	ON	Bound GET data items with colons in @ commands.
	OFF	Do not display colons as boundaries in @ commands.
SET CONFIRM	ON	**Do not skip to the next field in editing until a contol key (such as RETURN) is pressed.**
	OFF	Skip to the next field as each field is filled with characters.
SET CONSOLE	ON	Echo all output to the display screen.
	OFF	Do not echo output to the display screen.
SET DEBUG	ON	Send output from the ECHO and STEP commands to the printer.
	OFF	Do not send ECHO and STEP output to the printer.

Table 16-1 SET ON and SET OFF Commands (cont'd)

COMMAND		ACTION
SET ECHO	ON	Display all commands from a command file on the screen as the file is executed.
	OFF	Do not echo command file statements on the screen.
SET EJECT	ON	Eject one page before printing the output of a REPORT command.
	OFF	Suppress the page eject prior to printing REPORT command output.
SET ESCAPE	ON	Activate the **ESC** key, allowing the system user to abort execution of command files.
	OFF	Do not allow any escape from command files.
SET EXACT	ON	Require an exact match for character strings, except for trailing blanks, in expressions and the FIND command.
	OFF	Match character strings on the basis of the length of the second argument.
SET INTENSITY	ON	Use dual intensity for full-screen operations.
	OFF	Do not use dual intensity.
SET LINKAGE	ON	Make all sequential commands (LIST, SKIP, and so on) perform positioning on both the PRIMARY and SECONDARY data bases.
	OFF	Keep PRIMARY and SECONDARY data base operations independent.

Table 16-1 SET ON and SET OFF Commands (cont'd)

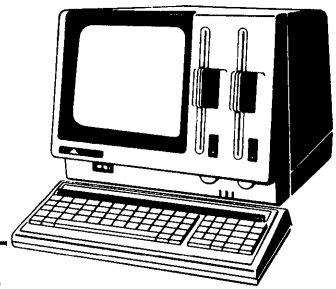
COMMAND		ACTION
SET PRINT	ON	Echo all output to the printer.
	OFF	Do not list output on the printer.
SET RAW	ON	Do not insert spaces between fields for LIST and DISPLAY commands that do not use a field list.
	OFF	Place spaces between fields for LIST and DISPLAY commands that do not use a field list.
SET SCREEN	ON	Use full-screen capability for EDIT, APPEND, INSERT, and CREATE.
	OFF	Turn off full-screen capabilities.
SET STEP	ON	Halt after the completion of each command in a command file. Wait for instructions to continue with the next command, escape from the command file, or accept a command from the keyboard. (STEP is used for debugging command files.)
	OFF	Follow normal operations.
SET TALK	ON	Display results from commands on the display screen.
	OFF	Do not display command results on the display screen.

Table 16-2 Character String SET Commands

COMMAND	ACTION
SET ALTERNATE TO [<file>]	This command is part of a two-step process to write everything that is normally written on the screen into a diskette file as well.
SET CALL TO <address>	This command is part of a two-step process to set up a branch to a jump vector or machine language subroutine. It specifies an address, in decimal, in the computer's memory.
SET DATE TO <mm/dd/yy>	The system date can be set or reset at any time with this command. It does not validate the date entered.
SET DEFAULT TO <disk drive>	This command makes the specified disk drive the default drive. dBASE II assumes that all files are on this disk drive unless the name is preceded by a different drive specifier. This allows you to write command files so that referenced files can be on any drive in the system. (You can also use the macro substitution function for disk drive assignment in command files.) When the default drive has been set, all inexplicit file names are set to the default. This is true for form files, command files, memory files, format files, index files, and text files, as well as data base files. The parameter <disk drive> may or may not have the colon (:) attached. That is, both "B" and "B:" are acceptable forms of specifying Drive B.

Table 16-2 Character String SET Commands (cont'd)

COMMAND	ACTION
	This SET command does not affect the CP/M default drive in any way. The dBASE II default drive is the same as the CP/M default drive you initiate a dBASE II session. The SET DEFAULT command redefines the system's internal default only within dBASE II.
SET FORMAT TO [SCREEN] [PRINT] [<format file>]	The first two formats of this SET command determine where the output of commands is sent. The last format names the source of input for READING data in the @ command.
SET HEADING TO <char string>	This command saves the character string internally and prints it as part of a REPORT header. The character string is saved for the duration of the current session only.
SET INDEX TO <index file list>	This command closes any index files that are currently open and then opens all index files named in the index file list. The first file named is called the master index. It is used to sequence the data base file. All index files listed are automatically updated with additions to and deletions from the data base in use.



Chapter 17

Introduction To Programming

For most applications, the commands covered to this point in the manual will be sufficient. You can create data bases, query them, and report the contents without creating programs. But if you want to, you can write programs to accomplish many functions using dBASE II. This chapter discusses the basic constructs, building blocks, from which computer programs are developed.

SEQUENCE

The command file you have already created demonstrates the first programming construct, *sequence*. This simply means that the system executes instructions in order, one after the other, from the first command it encounters to the last one in a file.

DECISIONS

Sometimes it is necessary to take different actions based on the data. Computers can make choices and decisions as long as they are presented in a way that the system can understand. Suppose you want to calculate raises for all employees in the EMPLOYEE file. The percentage of the increase is based on the current salary; 10% for those earning less than $15,000, 5% for everyone else. You could state the policy as a procedure in English something like this: If the salary is less than $15,000 add 10% to it, otherwise add 5%. Decisions are stated in dBASE II in much the same way. The salary increase policy could be written in the following form.

```
IF salary < 15000
        salary = salary + (.10 * salary)
ELSE
        salary = salary + (.05 * salary)
ENDIF
```

The IF...ELSE...ENDIF structure, or *decision construct*, is dBASE II's way of making choices. Try this structure in a command file to see how it works. Create a new command file called "RAISES" using **MODIFY COMMAND.** Then enter the file as it appears in the following display.

```
* RAISES 10-15-82  LAH
* Calculate and display raise
erase
set talk off
use employee
goto top
? "NUMBER:    " + str(number,3)
? "NAME:      " + last:name
? "OLD SAL:   " + str(salary,8,2)
IF salary < 15000
   store salary + (.10 * salary) to msalary
ELSE
   store salary + (.05 * salary) to msalary
ENDIF
? "NEW SAL:   " + str(msalary,8,2)
return
```

NOTE

Indenting is not required. However, it makes the command file easier to read.

Execute the file by entering:

do raises

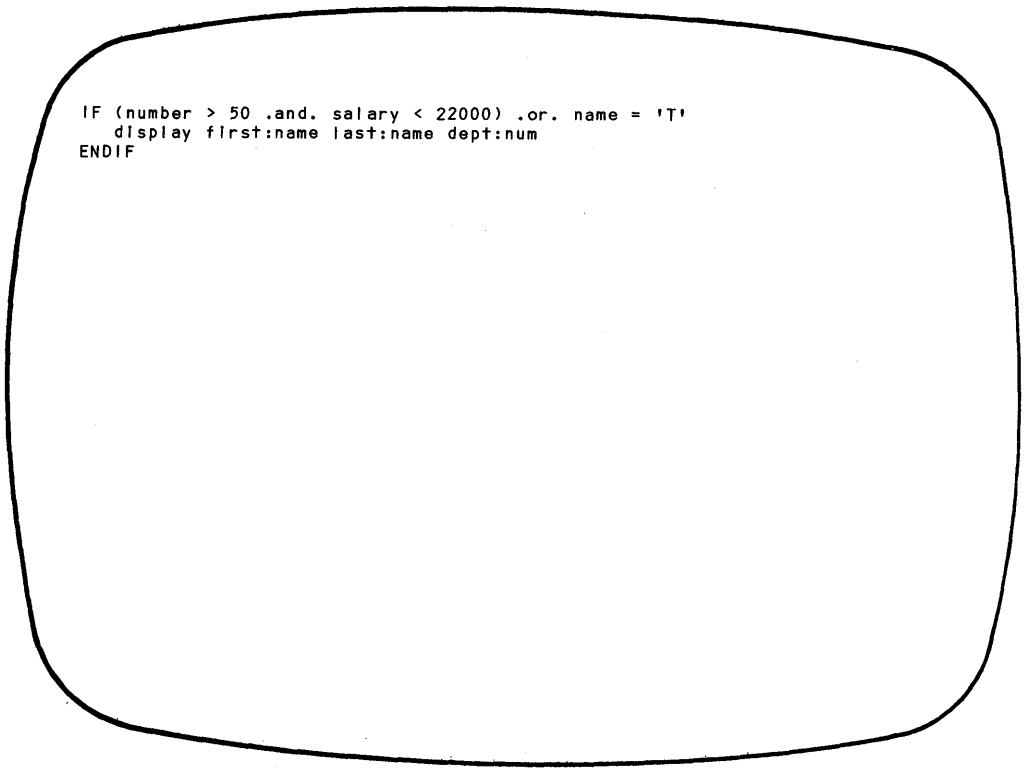
If the system displays the name, old salary, and correct new salary for the first employee, the command file worked. (If it did not execute properly, enter **MODIFY COMMAND RAISES** and check your command file against the example.)

The general form of the IF command is:

```
IF <exp>
   <statement>
[ELSE
  <statement>]
ENDIF
```

The IF command allows conditional execution of other commands. When <exp> is true, the commands following the IF are executed. When <exp> is false, the commands following the IF are bypassed and those following the ELSE are executed. ENDIF is required to signal that the IF command is complete.

The square brackets enclosing the ELSE clause indicate that it is optional. This is useful when you want to take certain actions if a condition is true and take no action if it is false. For example, the DEPTLIST command file created in Chapter 13 could be modified to print information about active employees only. The screen below demonstrates how the IF works with this kind of *simple decision.*



```
IF (number > 50 .and. salary < 22000) .or. name = 'T'
   display first:name last:name dept:num
ENDIF
```

In a simple decision, ELSE is not specified. The IF <exp> is evaluated. When it is true, all statements between IF and ENDIF are executed. When it is false, all commands between IF and ENDIF are skipped.

Notice that <exp> is complex. The IF condition can be a series of expressions (up to a maximum of 254 characters) that are evaluated logically. If *all* the conditions are true, the system performs the commands following the IF.

MULTIPLE CHOICES

Frequently, you have to make a choice and take an action from a list of alternatives. A common example is a screen menu from which a system user selects one of several different procedures.

Nesting

In creating a program that involves choices and selections, you can use a special form of the IF...ELSE ...ENDIF construction called "nesting".

```
IF <exp>
   <statement>
ELSE
      IF <exp>
         <statement>
      ELSE
         IF <exp>
            <statement>
         ELSE
            <statement>
         ENDIF
      ENDIF
ENDIF
```

Figure 17-1 Nested IF Command

Figure 17-1 demonstrates how much indenting helps you to understand nested commands. IF...ELSE...ENDIF commands may be nested to as many levels as necessary. Notice that every IF has a corresponding ENDIF.

NOTE

<statement> refers to whole command statements. The IF command begins with IF and ends with ENDIF. Statements must nest properly.

Case Structure

An alternative to nesting IF statements is the CASE structure. It can simplify coding by eliminating nesting. Figure 17-1 demonstrates a three-level nested IF. The same three levels are demonstrated using the CASE structure in Figure 17-2.

```
DO CASE
   CASE <exp>
        <statement>
   CASE <exp>
        <statement>
   CASE <exp>
        <statement>
   OTHERWISE
        <statement>
ENDCASE
```

Figure 17-2 Case Structure

The expressions can be any logical expression. Only the first CASE for which the expression is true is executed. Even if succeeding CASE expressions are true, processing skips to the first command after the ENDCASE command (see the programs in the sample application, Appendix H, for examples of the CASE structure).

REPEATING A PROCESS

RAISE works well for one employee, but the command should do the same thing for every employee on the file. The *repetition construct* is handled by the DO WHILE command in dBASE II.

```
DO  WHILE <exp>
          <statement>
ENDDO
```

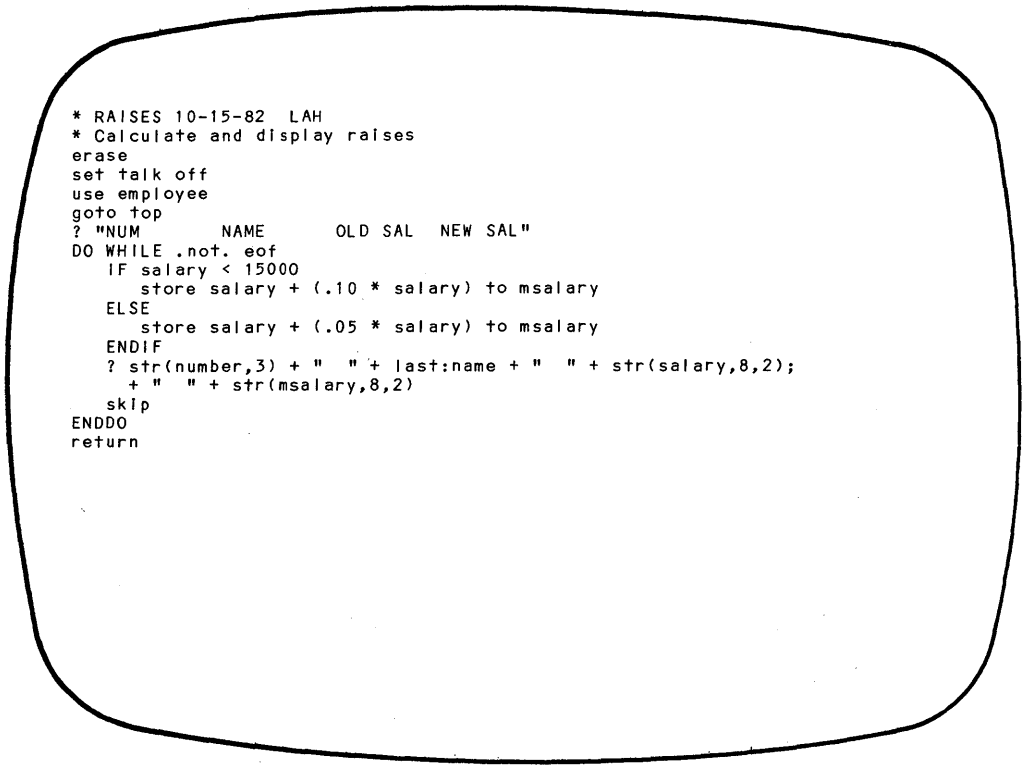
To make RAISES work for every employee record on the file, you can use the DO WHILE construction. Modify your RAISES command file to look like the one that follows. The DO WHILE command uses the end-of-file function. As long as the record pointer is not at the end of the file, the commands between DO WHILE and ENDDO are executed.

```
* RAISES 10-15-82  LAH
* Calculate and display raises
erase
set talk off
use employee
goto top
DO WHILE .not. eof
   ? "NUMBER:    " + str(number,3)
   ? "NAME:      " + last:name
   ? "OLD SAL:   " + str(salary,8,2)
   IF salary < 15000
      store salary + (.10 * salary) to msalary
   ELSE
      store salary + (.05 * salary) to msalary
   ENDIF
   ? "NEW SAL:   " + str(msalary,8,2)
   sk¶p
ENDDO
return
```

Now execute the command file by entering:

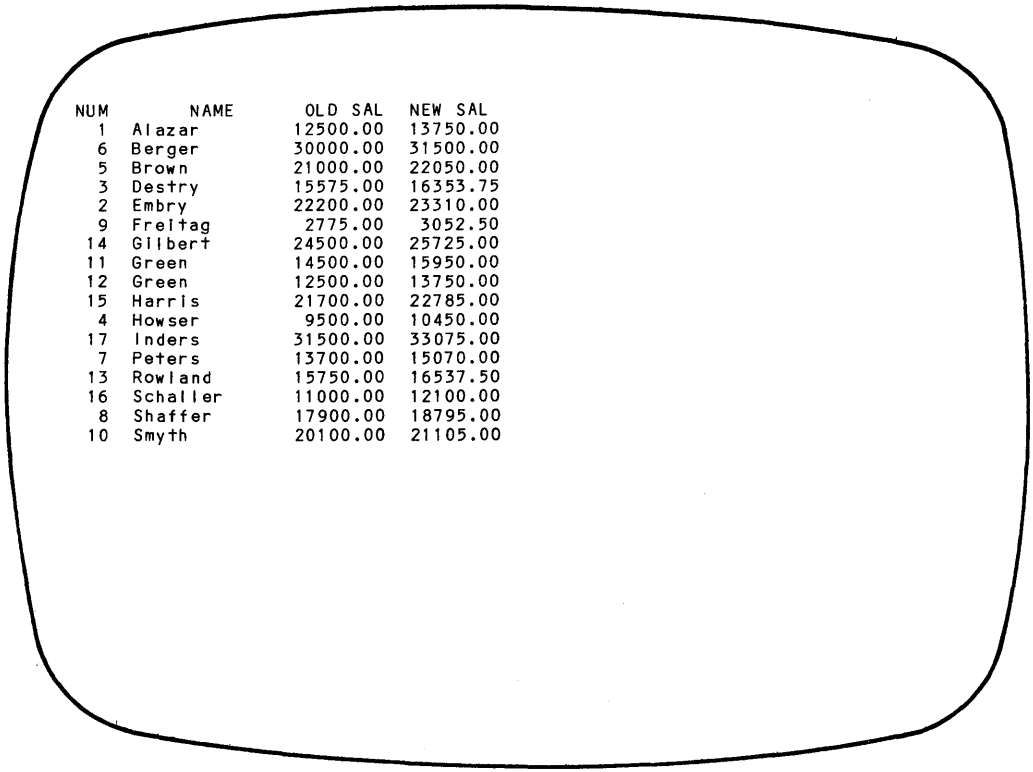
do raises

When the file is executed, the output scrolls by on the display screen so fast it is impossible to read. To correct this, you can add a line to the command file to turn the printer on. Then, the results can be printed. Alternatively, you can change the format of the output so that each employee's data is displayed on a single line. To do that, edit the command file as follows.



```
* RAISES 10-15-82  LAH
* Calculate and display raises
erase
set talk off
use employee
goto top
? "NUM       NAME       OLD SAL   NEW SAL"
DO WHILE .not. eof
   IF salary < 15000
      store salary + (.10 * salary) to msalary
   ELSE
      store salary + (.05 * salary) to msalary
   ENDIF
   ? str(number,3) + "  " + last:name + "  " + str(salary,8,2);
     + "  " + str(msalary,8,2)
   skip
ENDDO
return
```

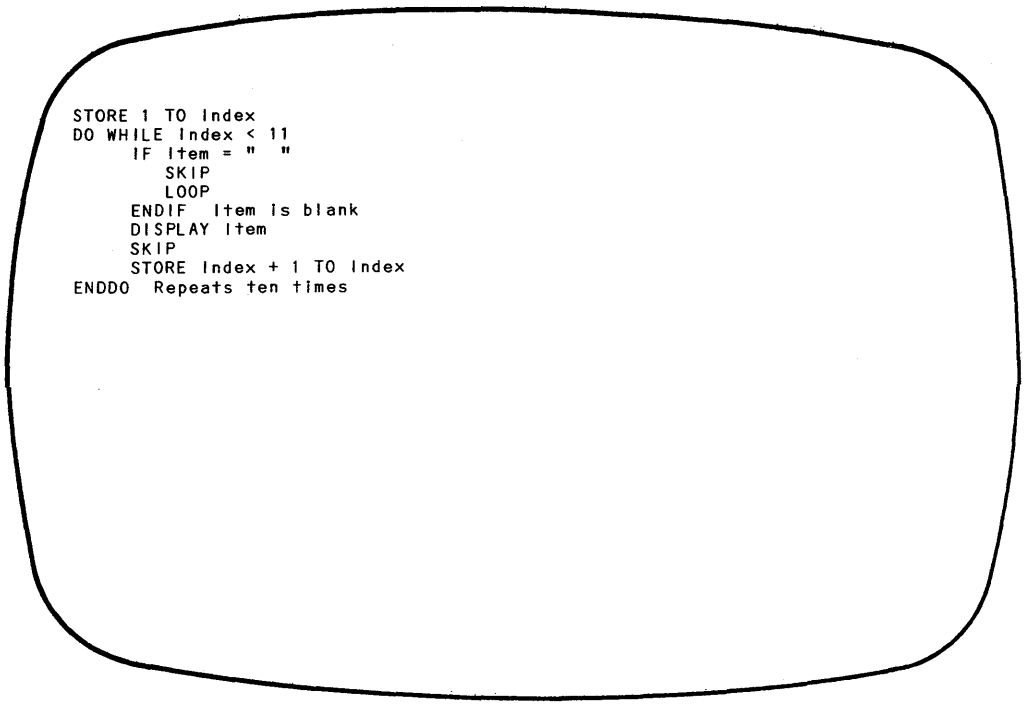
When the command file is executed, your screen should look like the display that follows.



```
NUM       NAME        OLD SAL   NEW SAL
  1   Alazar         12500.00  13750.00
  6   Berger         30000.00  31500.00
  5   Brown          21000.00  22050.00
  3   Destry         15575.00  16353.75
  2   Embry          22200.00  23310.00
  9   Freitag         2775.00   3052.50
 14   Gilbert        24500.00  25725.00
 11   Green          14500.00  15950.00
 12   Green          12500.00  13750.00
 15   Harris         21700.00  22785.00
  4   Howser          9500.00  10450.00
 17   Inders         31500.00  33075.00
  7   Peters         13700.00  15070.00
 13   Rowland        15750.00  16537.50
 16   Schaller       11000.00  12100.00
  8   Shaffer        17900.00  18795.00
 10   Smyth          20100.00  21105.00
```

Repetition is one of the major advantages of a computer. It can do the same task over and over without getting bored or making mistakes because of the monotony. However, the computer will continue to execute the commands within the loop forever if the value in the specified condition never changes.

DO WHILE constructs are useful when you know how many times you want a process repeated, as the sequence below demonstrates.



```
STORE 1 TO Index
DO WHILE Index < 11
     IF Item = "  "
        SKIP
        LOOP
     ENDIF  Item is blank
     DISPLAY Item
     SKIP
     STORE Index + 1 TO Index
ENDDO  Repeats ten times
```

In this example, the user wants to display the names of 10 items. The computer first stores the value "1" in a memory variable called INDEX. This is known as *initalizing the counter*. Then it enters the DO WHILE loop. The first thing that happens in this loop is a test to see if the instructions within it should be executed or bypassed. If the value of INDEX is less than 11, the computer proceeds through the DO WHILE instructions. When the counter passes 10, the computer skips the loop and performs the next instruction after the ENDDO. Within the DO WHILE loop, the system tests the value of ITEM. If there is data in the ITEM field, it is displayed, the value of INDEX is increased by 1, and the system skips to the next record. If the value of ITEM is spaces, the system just skips to the next record.

The LOOP instruction is used to stop a sequence and send the system back to the start of the DO WHILE loop that contains the instruction. In the example, if the ITEM field is blank, the record is not processed because the LOOP command moves the computer back to "DO WHILE INDEX < 11". The records with blank items are not counted, since this loop bypasses the instruction where the counter is incremented.

The problem with LOOP is that is short-circuits program flow, making it very difficult to follow program logic. The best practice is to avoid the LOOP instruction whenever possible.

COMBINING IFs and DO WHILEs

dBASE II statements in command files must nest correctly. To nest something means that one statement must fit completely inside another statement. This is especially important to proper execution of the IF-ELSE-ENDIF and the DO WHILE-ENDDO commands. Indenting a command file helps show that the statements are correctly nested. dBASE II does not catch nesting errors. It will, however, execute the command file in an unpredictable manner.

Figure 17-3 demonstrates the correct way to nest. The IF-ELSE-ENDIF statement is totally within the first DO WHILE-ENDDO command. The second DO WHILE statement is totally within the ELSE part of the IF-ELSE-ENDIF. It would be just as easy to show more levels of nesting, since dBASE II allows many more levels to exist.

```
DO WHILE .NOT. EOF
    .
    statements
    .
    IF A .AND. B
        .
        more statements
        .
    ELSE
        DO WHILE A <= 57
            .
            some more statements
            .
        ENDDO
        .
        more statements
        .
    ENDIF
    .
    more statements
    .
ENDDO
```

Figure 17-3 Correct Nesting

Figure 17-4 is an example of an error. The results of executing this file are unpredictable. The ENDDO crosses the boundary of the IF-ENDIF group. The statements do not nest properly. This command file would not execute properly and dBASE II would not explain why.

```
DO WHILE .NOT. EOF
    .
    statements
    .
    IF A .AND. B
        .
        statements
        .
    ENDDO
    .
    statements
    .
ENDIF
```

Figure 17-4 Incorrect Nesting

PROCEDURES

Command files can execute other command files. This means that you can create standard *procedures* and use them again and again. To call a command file from within a command file, use the DO command just as you would from the keyboard.

```
DO <command file>
```

DOs can be stacked up to 16 deep. That is, a command file can contain DO commands which invoke other command files that invoke other command files, and so on.

EXITING A COMMAND FILE

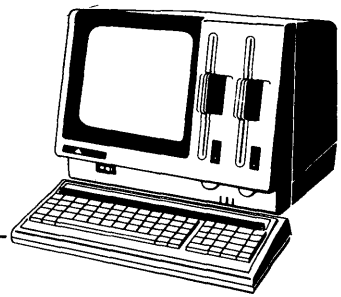
There are two ways to exit from a command file. The RETURN command signals that the end of the commands has been reached and execution of the file is complete. This command is used inside a command file to return control to the command file which called it or to the keyboard if the command file was called interactively. The command is the single word RETURN, without any modifiers. (It is not the RETURN key on the alphanumeric keyboard.)

CANCEL is used in command files to stop file execution and return to the normal keyboard interactive mode. The following segment of a command file demonstrates one use for this command.

```
IF status = 'S'
  cancel
ELSE
  display last:name status dept:num
ENDIF
```

A command file releases control when it reaches the end of the file or encounters the RETURN command. If the current command file was called by another command file, control is returned to the higher level command file.

If, during the execution of a command file, a CANCEL command is encountered, all command files are closed and the keyboard becomes the source for future commands.



Chapter 18

Interactive Programs

For some applications, command files need more information than is available in the data bases. The programs require communication and interaction with the system user. Command files can be set up so that they prompt the system operator with messages to indicate what information is needed. The commands presented in this chapter do this. They are useful in programs that present menus of functions from which one is to be selected. They are also found frequently in command files when it is important to insure that data is entered correctly.

ACCEPTING DATA INTERACTIVELY

The following commands all accept input from the keyboard. WAIT and ACCEPT take character data only. INPUT accepts all data types as input.

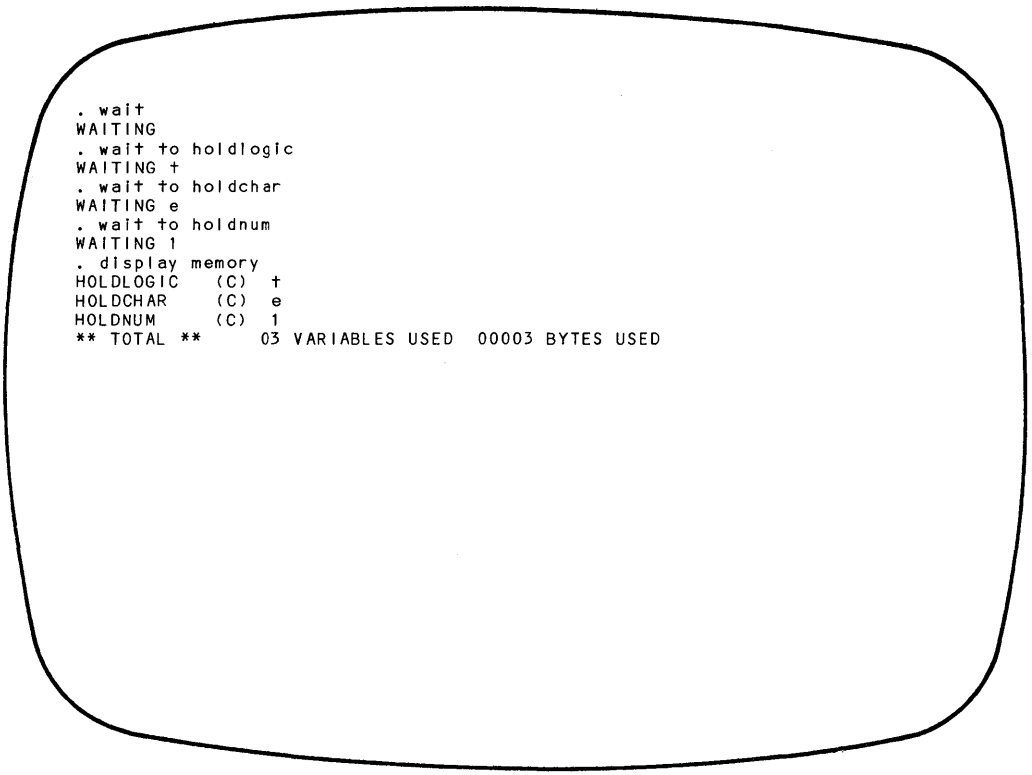
Wait Command

WAIT [TO <memvar>]

The WAIT statement halts command file processing. The system displays the prompt "WAITING" and waits for a single character to be typed at the keyboard. Processing continues after any key is pressed.

If a memory variable is specified in a TO phrase, the input character is stored in that variable. If the input is a non-printable character (**RETURN**, **ESC**, **CTRL**, and so on), a blank is stored in the memory variable.

WAIT can be used for rapid data entry since it reacts instantly to an input. The user does not have to press **RETURN** to enter the data. Therefore, the WAIT command should not be used when an incorrect entry could do serious damage to your data base.



```
. wait
WAITING
. wait to holdlogic
WAITING t
. wait to holdchar
WAITING e
. wait to holdnum
WAITING 1
. display memory
HOLDLOGIC    (C)  t
HOLDCHAR     (C)  e
HOLDNUM      (C)  1
** TOTAL **      03 VARIABLES USED  00003 BYTES USED
```

Input Command

INPUT ['<char string>'] TO <memvar>

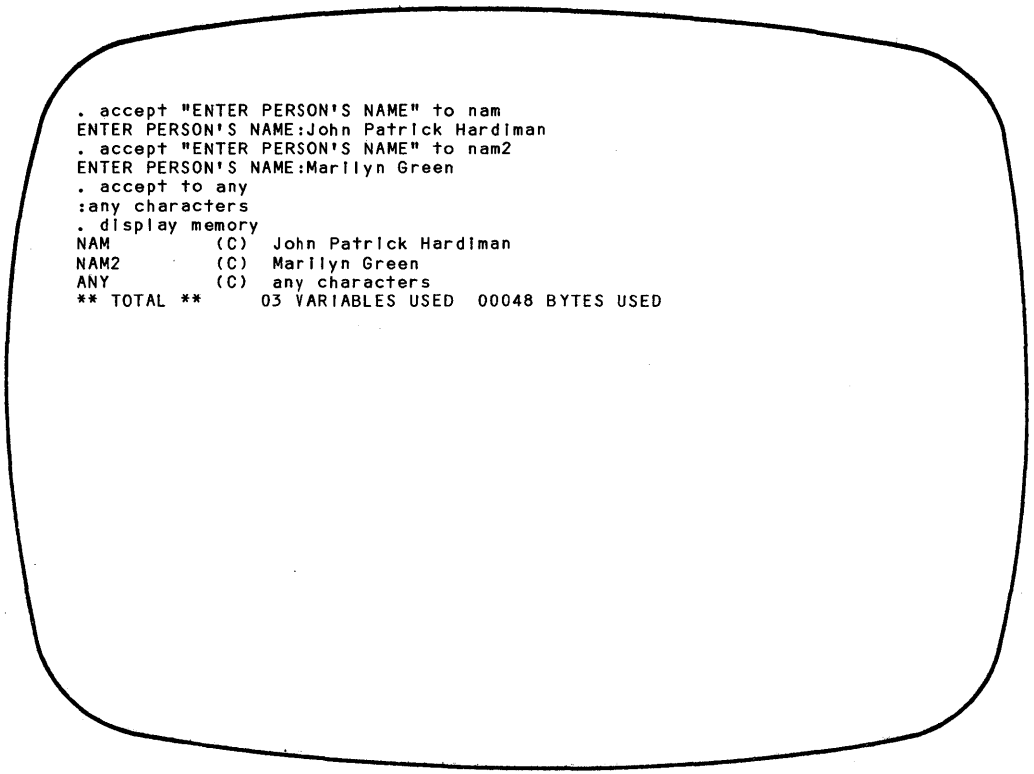
The INPUT command accepts any data type from the keyboard and stores the input in the named memory variable, creating that variable if it did not exist. If a character string is included, it appears on the display screen followed by a colon. The character string is a prompt. Data should be entered following the colon. The data type of the memory variable (character, numeric, or logical) is determined by the type of data that is entered. Character string prompts must be enclosed by quotes or square brackets.

```
. input to x
:3
  3
. input to z
:23/17.000 + x
 4.352
. input 'PROMPT USER TO FOR INPUT' to q
PROMPT USER FOR INPUT:12345
  12345
. input 'ENTER T IF EVERYTHING IS OKAY' to log
ENTER T IF EVERYTHING IS OKAY:t
.T.
. input "ENTER A CHARACTER STRING" to char
ENTER A CHARACTER STRING:'Character string must be delimited.'
Character string must be delimited.
. display memory
X           (N)    3
Z           (N)   4.352
Q           (N)    12345
LOG         (L)  .T.
CHAR        (C)  Character string must be delimited.
** TOTAL **      05 VARIABLES USED  00053 BYTES USED
```

Accept Command

ACCEPT ['<char string>'] TO <memvar>

The ACCEPT command accepts character data without the need for delimiters. It is useful for long input strings, where a forgotten quote mark can mean a lot of retyping. ACCEPT requires the operator to press **RETURN** after typing in the data. Therefore, it can also be used for single character entry when data integrity is important.



```
. accept "ENTER PERSON'S NAME" to nam
ENTER PERSON'S NAME:John Patrick Hardiman
. accept "ENTER PERSON'S NAME" to nam2
ENTER PERSON'S NAME:Marilyn Green
. accept to any
:any characters
. display memory
NAM          (C)  John Patrick Hardiman
NAM2         (C)  Marilyn Green
ANY          (C)  any characters
** TOTAL **      03 VARIABLES USED  00048 BYTES USED
```

FORMATTING THE SCREEN

ACCEPT, INPUT, and the ? command can all be used to place messages and prompts to the system user on the screen. The problem with all of these commands, however, is that the character strings appear just below the last line of print that is already on the screen. There is a better way to ask for input.

Positioning the Cursor

The @ command positions prompts and accepts data from almost any position on the screen. At its simplest, the format of the @ command is:

@ <coordinates> [SAY '<char string>']

The @ command positions the character string prompt, if any, at the screen coordinates specified. The coordinates are the *row* and *column* on the screen. Position 0,0 is the top row, first column; position 23,79 is the bottom row, last column. If 9,34 were specified as the coordinates, the prompt would start on the tenth row in the 35th column.

Displaying Prompts and Messages

The SAY phrase is optional. Without it, the @ command erases the row on the display screen indicated by the row coordinate beginning with the column specified by the column coordinate.

To see how the command works, enter the following commands at the prompt:

```
erase
@ 20,30 say 'What?'
@ 5,67 say 'Here...'
@ 11,11 say "That's all."
@ 20,0
@ 5,0
@ 11,16
```

Displaying Data Field Values

Instead of only displaying a prompt, the command can be used to show the value of an expression consisting of one or more variables.

@ coordinates [SAY <exp>]

Enter the following commands to test this format:

```
erase
use employee
@ 13,9 say zip:code
@ 13,6 say state
skip 3
@ 23,5 say last:name +',' + street
```

The @ command can be expanded further to show the values of variables at whatever screen position is specified.

@ <coordinates> [SAY <exp>] [GET <var>]

To see how this works, enter the following commands:

```
erase
use employee
@ 15,5 say 'state' get state
@ 10,17 get zip:code
@ 5,0 say 'Name' get last:name
```

This sequence positions the values of the variables, with and without prompts, at various places on the screen. With this facility, you can design displays so that your screens look just like the paper forms that they are replacing.

@ commands do not have to be presented in row order when writing to the screen, but it is good practice to write them this way.

- The top row of the screen, row 0, is reserved for special dBASE II purposes. Therefore, do not issue a format command for that row.
- The @ command with no GET or SAY phrase clears a line or the part of a line, from the designated column to the end of the line. For example, the command "@ 10,12" erases row 10 from column 12 through the end of the row.
- The coordinates can be literals, variables, or expressions.
- When the GET phrase is used without the SAY phrase, only the colons delimiting the field length for the variable are displayed.

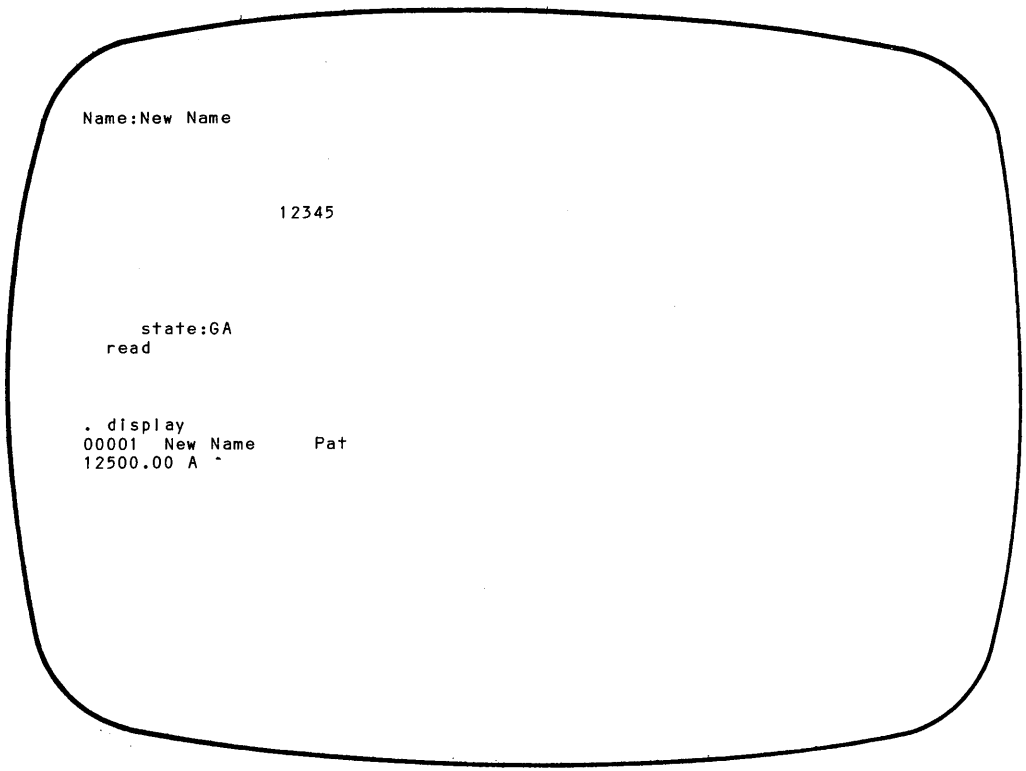
FILLING IN THE DATA

Once the screen has been formatted using @ commands, the system is ready to accept data. To get data into the variables on the screen, enter:

read

The cursor is positioned on the first field displayed by an @ command. The operator can type in new data or leave the field as it is by pressing **RETURN**. When data fills a field or **RETURN** is pressed, the cursor jumps to the next variable to accept data.

In the example that follows, the state, zip code and name fields were edited and the results were stored in the data base file. You can verify this by executing the display command after editing the data.



READ is a one-word command. It tells the system to enter full-screen edit mode for editing and/or data entry. Additions and changes made to the variables on the screen are entered into the appropriate data base fields or memory variables. The SET SCREEN ON command must be in effect for this command to operate. This is the default condition for that SET command.

Variables used with the @ command and edited with READ must either be in the file that is in USE as field names or be in the system as character string memory variables. Memory variables must be defined prior to the @ command. If necessary, set up the memory variable by creating it with as many blanks as the maximum number of positions the field should occupy. (For example, "STORE ' ' TO MEMVAR" sets up a variable called MEMVAR with a length of five.)

You must use the ERASE or CLEAR GETS command after every 64 GET commands are issued. CLEAR GETS removes all pending GETs. All READ commands that follow will READ only the input from GETs that are issued after the CLEAR GETS. ERASE accomplishes the same action of removing all pending GETs and also clears the display screen.

If the SET FORMAT TO <format file> command has been issued, READ causes all of the @ commands in the format file to be executed. This sequence formats the screen and allows editing of all GET variables.

If the SET FORMAT TO SCREEN command has been issued, issue an ERASE command to clear the screen first. Then, use a series of @ commands to format the screen. To edit the fields, use the READ command. If a second or later series of @ commands is issued after a READ command, READ places the cursor on the first GET variable following the last READ. In this way, the screen format and the specific variables edited can be based on decisions made by the user in response to prior READ commands.

FORMATTING THE INPUT FIELDS

The @ command can be expanded still further for special formatting.

@ <coordinates> [SAY <exp>] [GET <var>] [PICTURE '<format>']

The optional PICTURE phrase is filled in using the format symbols listed in Table 18-1.

Table 18-1 PICTURE Symbols for @ Command

SYMBOL	ACTION
9 or #	Accept only digits as entries
A	Accept only alphabetic characters
!	Convert character input to uppercase
X	Accept any characters
$	Show '$' on the screen
*	Show '*' on the screen.

The following screen demonstrates how to enter the picture symbols in @ commands. To test their effect on data entry, enter the commands on your system and type in data values.

```
. erase
. store '00000' to digits
00000
. store '     ' to letters

. store '     ' to convert

. store '     ' to amount

. store '     ' to stars

. erase
. @ 2,5 say "digits " get digits picture '99999'
    digits :00000:
. @ 4,5 say "letters " get letters picture 'AAAAA'
    letters :     :
. @ 6,5 say "convert " get convert picture '!!!!!'
    convert :     :
. @ 8,5 say "amount " get amount picture '$9.99'
    amount :$ .  :
. @ 10,5 say "starfill " get stars picture '**999'
    starfill :**   :
. read
```

PRINTING FORMS

If you SET FORMAT TO PRINT, the @ command formats the prompt on the printer instead of the display screen. The GET and PICTURE phrases are ignored, and the READ command cannot be used. The @ commands must be in order by the row and column coordinates so they can be printed from top to bottom, left to right. Use this SET option with the @ command in command files to produce forms.

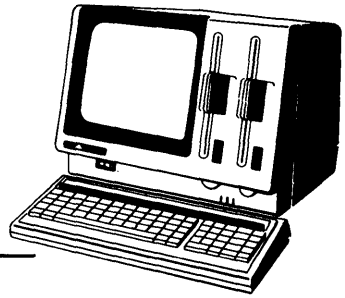
A SAMPLE DATA ENTRY PROCESS

A data base may have many fields but for any given data entry procedure, you may be entering data in only some of them. Rather than using APPEND, which lists all the fields in the data base on the screen, you can use APPEND BLANK to create a record with empty fields and then GET only the data you want.

To give you more practice with command files, create a command file called TRIAL.PRG consisting of the program displayed on the following screen.

```
erase
? 'This procedure allows you to add new records to the '
? 'EMPLOYEE.DBF file selectively.  You will add only '
? 'last:name and zip:code now.'
?
? 'Press S to stop the procedure.'
? 'Press RETURN to continue.'
wait to continue
use employee
do while continue <> 'S' .and. continue <> 's'
     append blank
     erase
     @ 10,0 say "LAST NAME " get last:name
     @ 10,30 say "ZIP CODE " get zip:code
     read
     ? '  S to stop the procedure,'
     ? '  RETURN to continue.'
     wait to continue
enddo
return
```

DO the command file to see that it works. Enter several records. After you have finished, LIST the file to check what was added. As you can see, the data entry screen is simple and uncluttered. The screen can be customized by placing prompts and variable input fields where you want them.



Chapter 19

Planning Command Files

The first thing to do when you want to write a command file is turn the computer off. That is where many programmers go wrong. They immediately start "coding" a solution, before they even have a clear idea of what problem they are solving. Don't worry. You will have plenty of opportunities to use all the commands and functions you have read about. But first, take the time to plan how to use them.

This chapter uses part of the sample application (see Appendix H) to demonstrate the recommended approach for planning command files.

DEFINING THE PROBLEM

Start by defining the problem in ordinary English. Make it a general statement, something like this: The purpose of this program is to update the EMPLOYEE file with additions, changes, and deletions.

Now define the problem further. What inputs does the program need? In what form do you want the outputs and reports? For the sample program, the inputs are the EMPLOYEE file and data that the operator enters from the keyboard. The operator will be presented with a menu with the following options: Add New Employee, Change Current Employee Information, Delete Employee Record. Depending on the option selected, the appropriate action will be taken. When all changes have been made, the operator should get a list of all updates made to the file during the session and a count of additions, changes, and deletions.

HANDLING EXCEPTIONS

Once the normal processing flow is sketched out, think about the exceptions. What are the starting conditions? What happens if a record is missing? This program will have to check that additions and changes have valid job and department codes and that the salary is within the range for the job code. It will have to handle errors such as the operator trying to change or delete the record of an employee who is not on the file.

DETERMINING THE FLOW OF PROCESSING

After you have defined what the program should do, describe the details. Use English terms that are somewhat similar to the instructions that dBASE II understands: IF...ELSE, DO WHILE, FOR, and so on. In the sample application, the first part of the data entry procedure is the presentation of the menu and the selection of a function. The following outline describes that sequence of actions in an English-like form.

```
Use the EMPLOYEE file
Present the menu and get a selection
If the choice is Add New Employee
    Do the append routine
If the choice is Delete Employee
    Do the delete record routine
If the choice is Change Employee Data
    Do the change routine
For each modification to the file
    Display the record after any processing
    Keep a running count of the number of additions,
    deletions and changes
```

After you finish the first outline, go back to the beginning and add more detail, name procedures, and reorganize the processing as necessary. After a few revisions, the program might look like the next outline. It is still not a program that dBASE II can execute, but many of the commands are starting to take shape, gaps have been filled in, and potential problems are identified.

```
USE EMPLOYEE
STORE 0 too numadd, numdel, numchng
STORE ' ' TO choice
*Need to have two files open - remember to select primary
*and secondary as necessary.
*EMPLOYEE will be primary. TEMPFILE will be secondary.
DELETE all the records in tempfile
PACK or COPY to tempfil2
DELETE tempfile
RENAME tempfil2 to tempfile
DO WHILE choice .NOT. "E"
    DO MENU
    IF choice = 'A'
```

```
            DO ADD
        ELSE
          IF choice = "D"
              DO DELETE
            ELSE
              IF choice = "C"
                DO CHANGE
              ELSE
              ? "TRY AGAIN. PRESS EITHER A, D, C, or E."
              ENDIF
            ENDIF
        ENDIF
    ENDDO
    LIST tempfile
    DISPLAY numadd, numdel, numchng
```

TESTING COMMAND FILES

Start over again at the top and continue adding detail, including comments, until the dBASE II commands are in a format that you can begin testing. With this approach, the entire system does not have to be written before you begin testing. You can save the complicated details until late in the program development process. In fact, you are probably better off not worrying about details too early on because then you can concentrate on the overall problem solution. Go back after you have tested the overall solution and clean up the procedures then.

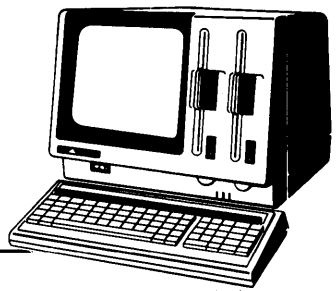
You can test a partial program that calls other command files (ADD, CHANGE, DELETE, and MENU in this example) by creating dummy files. The dummy files consist of a single instruction or a few instructions, enough to let you know that the procedure was called. For instance, to test the way the program calls ADD.PRG based on operator input, ADD.PRG can consist of the following two instructions.

```
? "COMMAND FILE ADD"
RETURN
```

When you test the program and see "COMMAND FILE ADD" displayed on the screen, you know that the command file named "ADD.PRG" was executed.

In computer programming texts, the terms top-down, step-wise refinement are applied to this kind of program development procedure. Actually, it's just a sensible approach to solving most kinds of problems. First, state the overall problem, trying

to define what it is and what it is not. Gradually get into more and more detail. Begin testing the solution as soon as enough of the program is finished to give some results. Just as you develop command files in pieces, test each part separately and then test again to be sure each part integrates with the whole system.



Chapter 20

CP/M Interfaces

INTERFACING WITH CP/M DATA FILES

dBASE II can read data from files that were created by text editors, word processors, and other programs and can generate files that are compatible with other programs and systems. This interface is possible with files that are in ASCII text format using the CP/M convention of a line of text followed by a carriage return and line feed.

Appending Data From CP/M Files

The APPEND command, in the following format, reads ASCII text files created by dBASE II and other systems.

APPEND FROM <file> [FOR <exp>] [SDF] [DELIMITED WITH <delimiter>]

If SDF is specified, dBASE II assumes that the records are in Standard Data Format (ASCII text format). If the FROM file records are smaller than the records in the USE file, the new records are padded on the right with blanks. If the FROM file records are longer than the USE file records, the newly appended records are truncated. Records are added to the USE file until end-of-file is detected in the FROM file.

Some processors add field delimiters to the records. A popular format uses commas to separate fields and single or double quotes to enclose character strings. dBASE II can accept records in these formats when the DELIMITED phrase is specified. dBASE II removes the quotes and commas from a DELIMITED file and stores the data in a dBASE-structured data base file according to the data base's structure.

Creating CP/M Compatible Output Files

The COPY command can create delimited and non-delimited ASCII text files with the following command.

COPY TO <file> [<scope>] [FIELD <field list>] [FOR <exp>] [SDF]
[DELIMITED [WITH <delimiter>]]

If SDF is specified, the file in USE is copied to the named file without the structure. It is in standard ASCII format. The extension ".TXT" is automatically appended unless another extension is specified.

If the DELIMITED keyword is used, the output file will have all of its character string fields enclosed in quotes and the fields will be separated by commas. By default, COPY DELIMITED uses single quotes as delimiters to mark character string fields. The WITH sub-phrase of the DELIMITED phrase allows any character to be the delimiter. If a comma (,) is used as the delimiter, the character fields will have trailing blanks trimmed, the numeric fields will have the leading blanks trimmed, and the character strings will not be enclosed in quotes.

WRITING TO DISKETTE

When the system is in command mode, everything that is written on the display screen or printed on the printer can also be written to a disk file. This includes the system output as well as commands and other inputs typed at the console. The diskette file that contains screen and printer text is created in standard ASCII format using the CP/M convention of a line of text followed by a carriage return and line feed. The file may be saved as an audit trail for a terminal session, printed, or used as input to a word processor, text editor, or other program. (The screens in this manual were created this way and then modified using a word processor.)

Use the following procedure to copy command mode text to diskette:

1. If you have a two-drive APC and want to create the text file on Drive B, insert a formatted diskette in Drive B.
2. Identify the text file to the system by entering the following command at the dot prompt.

   set alternate to [<disk drive>:]<file>

   If the file does not exist, the system automatically creates the file. If a file with the same name already exists, its contents will be overwritten.

3. To begin the echo process, enter the following command at the dot prompt.

   set alternate on

   From now on, until you enter a command to set alternate off or exit from dBASE II, everything that is entered, displayed, and printed in command mode is echoed to the text file.

CHAINING dBASE II TO OTHER PROGRAMS

When a dBASE II session is terminated, you can exit back to the CP/M-86 operating system or the system can automatically begin executing other programs. An optional phrase for the QUIT command initiates the process of chaining to other command files.

QUIT [TO<CMD file list>**]**

The programs in the <CMD file list> are executed in sequence by CP/M-86. You can reenter dBASE II after the other programs have finished executing by entering **"DBASE**<CMD file>**"**, but it is not required. If dBASE II is not the last program in the command file list, the CP/M-86 operating system will be given control when all command files have been executed.

There is no limit to the number of CP/M commands that can be executed, but the normal dBASE II restriction of 254 characters per command line must be followed. The command file names must be enclosed in single quotes and separated by commas.

RESETTING THE SYSTEM

The RESET command is used to reset the CP/M bit map after a diskette has been swapped. Normally, if a diskette is swapped, CP/M will not allow writes to take place until after a soft boot. RESET attempts to reopen all files that were open prior to the swap. If a file that was open is no longer mounted on an active disk drive, RESET closes the file internally.

Issuing a RESET command when no disk swap has taken place has no effect.

WARNING

> If the new disk contains a file with the same as a name file that was previously open, the RESET operation will erroneously not close that file. This can be avoided if you close all non-essential files prior to the swap and subsequent RESET command. A USE command with no file name closes the file in USE; a CANCEL command closes any open command files.

ACCESSING THE APC'S MEMORY BY LOCATION

dBASE II allows access to specific APC memory locations with four operations: PEEK, POKE, SET CALL TO, and CALL. To use these commands and functions, you should be familiar with the organization of the computer's memory, the CP/M-86 operating system, and assembly language commands and subroutines. (See the *CP/M-86 System Reference Guide* for the APC for more information about these topics.)

Finding A Byte Value

PEEK is a numeric function that returns the value of a specified byte in the computer's memory. It is useful for operations like testing the value of a port for the status of an I/O device. The format of the function is:

PEEK (<address>)

The address is a decimal value that specifies a byte location. The function returns the contents of the address in decimal format.

Storing Values

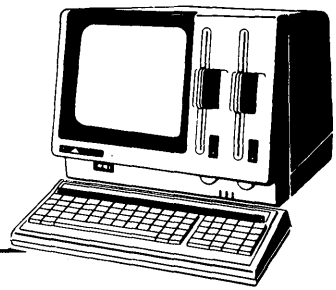
POKE stores a list of values into the computer's memory starting at the specified location. The command format is:

POKE <address>, <byte list>

The address and byte list values are expressed in decimal. Values in the byte list must be separated by commas. This command is used for such functions as resetting I/O device status bytes and storing subroutine instructions.

Executing Assembler Subroutines

The SET CALL TO <address> command specifies the decimal address of a jump vector or subroutine. CALL <memvar> can then be used to branch processing to the address specified in the SET command. The memory variable must be a string. dBASE II saves the registers upon entering the subroutine and restores them upon exit. The subroutine must execute a machine language return to get back to dBASE II. Use this sequence of commands to enter the APC BIOS and to execute machine language subroutines.



Appendix A

Full Screen Edit Commands

KEY(S)	ACTION	EDIT	BROWSE	APPEND CREATE INSERT	MODIFY STRUCTURE	MODIFY COMMAND
TAB ↓ **CTRL F** **CTRL J**	Move ahead one field	x	x	x	x	
SHIFT TAB ↑ **CTRL E** **CTRL A** **CTRL K**	Move back one field	1	1	x	x	
---→ **CTRL D**	Move ahead one character	x	x	x	x	x
←--- **CTRL S**	Move back one character within a field and back one field from first character of a field	x	x	x	x	x
DEL **CTRL G** **CTRL X**	Delete character under cursor	x	x	x	x	x
CTRL C	Save changes and advance to next record	x	x	x	2	3
CTRL R	Save changes and back up one record	x	x		4	5

KEY(S)	ACTION	EDIT	BROWSE	APPEND CREATE INSERT	MODIFY STRUCTURE	MODIFY COMMAND
CTRL W	Exit, save changes on screen	x	x	x	x	x
CTRL Q	Exit without saving current changes	x	x	x	x	x
CTRL Z	Pan window one field to the right		x			
CTRL B	Pan window one field to the left		x			
CTRL T	Delete line under cursor				x	x
CTRL N	Insert blank line				x	x
CTRL Y	Erase field	x	x	x	x	
CTRL U	Toggle record deletion mark	x	x	x		
INS CTRL V	Toggle insert/overtype	x	x	x	x	x
PRINT CTRL P	Toggle printer on/off	x	x	x	x	x

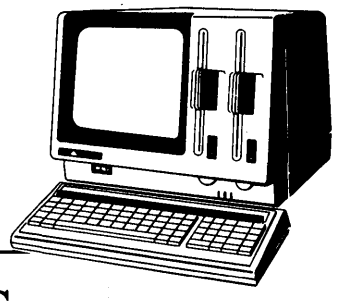
1 — When the cursor is on the first character of the first field in a record, this command moves the cursor back to the first character in the first field of the previous record.

2 — Advance to the next panel of fields.

3 — Advance one screen.

4 — Move back to the previous panel of fields.

5 — Move back to the previous screen.



Appendix B

dBASE II Command Symbols

Table B-1 defines the symbols used in dBASE II commands. Understanding the special symbols used in the general formats of the commands is vitally important. Not only does it help in understanding just what the format of the command really is, but it helps to show the potential of the command.

Table B-1 Command Symbols

SYMBOL	MEANING
<command file>	Name of a command file with the "PRG" extension.
<char string>	Character string. Must be enclosed in single quotes, ('), double quotes, ("), or square brackets, ([]).
<delimiter>	Any special character, including " () * = , @.
<exp>	Expression. Can be created by combining numbers, functions, field names, or character strings in any meaningful manner. For example 4+8, doc = '3' .or. doc = '4', and "$('abc'+&somestr,n,3)='abcdefg' " are all expressions.
<exp list>	one or more expressions, separated by commas.
<field>	Any record field name.
<field list>	One or more field names, separated by commas.
<file>	Any file name.
<form file>	Name of a report form file with the "FRM" extension.

Table B-1 Command Symbols (cont'd)

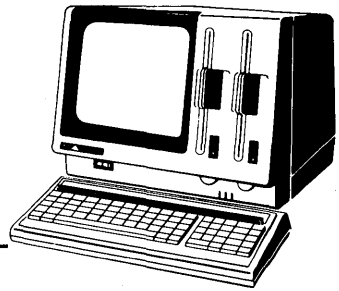
SYMBOL	MEANING
<format file>	Name of a format file with the "TXT" extension.
<index file>	Name of an index file with the "NDX" extension.
<key>	The field name on which a file is indexed. Keys may be variable names or expressions.
<mem file>	Name of a memory file with the "MEM" extension.
<memvar>	Memory variable.
<memvar list>	One or more memory variables, separated by commas.
<n>	Numeric literal.
<scope>	How much of the file a command covers. Acceptable values are: ALL NEXT n RECORD n
<statements>	Any valid dBASE II commands. Must be *whole* statements.
<type>	Three-character file type extension.

Table B-1 Command Symbols (cont'd)

SYMBOL	MEANING
<var>	Variable name.
FOR <exp>	Any record so long as some logical expression has a true value. Unless otherwise specified, the presence of a FOR phrase causes ALL records to be scanned starting with the first record in the file.
WHILE <exp>	All sequential records as long as some logical expression has a true value. The controlling command stops the first time the expression is false. The presence of a WHILE clause implies NEXT 65534 unless otherwise specified. It starts at the current position.

NOTE

There are other special symbols used in the command formats. These are specific to the command and are explained in the body of the command.



Appendix C

Summary of dBASE II Commands

This appendix is an alphabetical listing of the formats of dBASE II commands.

TYPOGRAPHIC CONVENTIONS

The following conventions are used in the command formats.

[...]	Square brackets indicate parts of commands that are optional. DO NOT TYPE THE SYMBOLS.
<...>	Pointed brackets enclose portions of commands that are to be filled in with real information. DO NOT TYPE EITHER THE SYMBOLS OR THE LOWERCASE WORDS AND ABBREVIATIONS THEY ENCLOSE. Replace them entirely with your information.
/	Slash indicates alternatives. Select one and only one.
"..."	In certain commands, character strings must be enclosed within delimiters. The double quotes in the command summary identify these. You must use delimiters: matching pairs of double quotes, single quotes, or square brackets.
UPPERCASE	This type style indicates verbs and keywords. They may be abbreviated to four characters or more and must be spelled as presented in the command format. You may enter commands in uppercase and/or lowercase.

ALPHABETICAL SUMMARY OF dBASE II COMMANDS

* [<char string>]

Places comments into a command file. <char string> is not displayed when the file is executed.

? [<exp list>]
?? [<exp list>]

Displays the value of one or more expressions. The expression list is optional. A "?" with no expression spaces down one line on the output. "??" operates exactly like "?" except that the result is printed on the same line as the command.

@ <coordinates> [SAY <exp> [USING "<format>"]]
[GET <var> [PICTURE "<format>"]]

Displays formatted information on the screen or printer. The <coordinates> are a row, column pair. The display screen is 24 rows by 80 columns numbered as rows 0 -23 and columns 0 - 79. The 0th row is reserved for system messages and should not be used. For the printer, both coordinates have ranges of 0 - 254. The coordinates can be literals, numeric memory variables, or numeric expressions. With no optional phrases, this command clears a row, or part of a row, on the display screen to spaces beginning with the row,column coordinates. The <format> consists of characters that have special meanings to dBASE II to identify character and numeric input, dollar signs, and so on (see Chapter 18).

ACCEPT ["<char string>"] TO <memvar>

Stores keyboard input to a character string memory variable. The memory variable is created if it does not already exist. If "<char string>" is present, it is displayed as a prompt message followed by a colon. If it is not present, a colon is displayed as the prompt. Keyed input is accepted as character type. It does not have to be enclosed within quotes.

APPEND
APPEND BLANK
APPEND FROM <file> [FOR <exp>] [SDF] [DELIMITED WITH <delimiter>]

In all three forms, records are added to the end of the data base in USE. In the first form, the user is prompted with the field names from the USE file's structure. In the second form, a single, space-filled record is appended. In the third form, records are taken from another file. If SDF is present, the records are assumed to be in Standard Data Format. If the DELIMITED phrase is included, the records taken from the FROM file are assumed to be delimited. If the SDF and DELIMITED phrases are not present, the FROM file is assumed to be a dBASE II-structured data base file.

BROWSE

Displays data from up to 19 records at a time for editing and viewing. As many fields as will fit are put on each line. They can be edited with standard editing commands.

CALL <memvar>

Branches to the address specified in a previous SET CALL TO command.

CANCEL

Aborts command file execution.

CHANGE [<scope>] FIELD <field list> [FOR <exp>]

Makes multiple changes to a data base. All data base fields that are referenced in the list are presented in the order given by <field list>. They may be edited or skipped.

CLEAR [GETS]

Resets dBASE II, closing all files, clearing all memory variables, and reselecting the primary work area. If the GETS keyword is used, clears all the pending GETs and leaves screen intact.

CONTINUE

Used with the LOCATE command to search for another record. LOCATE and CONTINUE may be separated by other commands.

COPY TO <file> [<scope>] [FIELD <field list>] [FOR <exp>]
[DELIMITED [WITH <delimiter>]] [SDF]

Copies the USE data base to another file, creating that file if it does not already exist. If SDF is *not* specified, the structure and all records not marked for deletion are copied. If SDF is specified, only the data records are copied in ASCII standard format. If the DELIMITED keyword is used, the character string fields are enclosed in single quotes and all fields are separated with commas. The WITH sub-phrase allows the user to specify any character as the delimiter. If "," is used as the delimiter, character fields have trailing blanks trimmed, numeric fields have leading blanks trimmed, and character strings are not enclosed in quotes. If either the DELIMITED or SDF option is used, the output file name defaults to the "TXT" extension.

COPY STRUCTURE TO <file> [FIELD <field list>]

Copies only the structure of the USE data base to another file, creating that file if it does not already exist.

COUNT [<scope>] [FOR <exp>] [TO <memvar>]

Counts the number of records in the USE file, including records marked for deletion.

CREATE [<file>]

Creates a new data base file. Prompts user for name if it is omitted.

DELETE [<scope>] [FOR <exp>]

Places deletion mark (*) in first character of all records that are within <scope> and which satisfy the FOR expression. Default is current record.

DELETE FILE [<disk drive>:]<file>

Deletes the named file from the CP/M directory on diskette and releases the space it occupied to the operating system for reassignment. If the <disk drive> is omitted, the default drive is assumed.

DISPLAY [<scope>] [FOR <exp>] [<field list>] [OFF]

Displays all or part of the data base file in USE. If <scope> and FOR are not present, only the current record can contribute information for the display. If a FOR phrase is present, the default is ALL. OFF inhibits displaying the record sequence number.

DISPLAY FILES [ON <disk drive>] [LIKE *.<type>]

Displays names of files residing on a diskette. If there are no phrases, DBF files are listed. ON allows the specification of a drive other than the default drive. The LIKE phrase allows files of type other than DBF to be displayed. The <type> is the three character extension such as TXT, FRM, MEM, and so on.

DISPLAY MEMORY

Displays all currently defined memory variables. Gives name, type, length, and contents for each variable and total number of bytes for all memory variables.

DISPLAY STRUCTURE

Displays the structure of the USE data base. Lists name, type, length for each field, total record length, and usage such as date of last update and primary/secondary area assignment.

DO <command file>

Executes the commands in the named file. When dBASE II reaches a RETURN command or an end-of-file indicator, it returns control to the source of the DO command (either another command file or the keyboard).

```
DO CASE
    CASE <exp>
            <statements>
    CASE <exp>
            <statements>
    .
    .
    .
    [OTHERWISE]
            <statements>
ENDCASE
```

Extends the DO command to execute one of a list of processes depending on the logical value of <exp>. dBASE II executes DO CASE as if it were a list of IF-ENDIFs. It examines the expressions in the individual CASEs and executes the statements following the first expression that evaluates as TRUE. When it reaches the next phrase beginning with CASE, it exits to the ENDCASE.

If the OTHERWISE clause is present and none of the CASEs is true, the statements in the OTHERWISE clause are executed.

Any statements placed between the DO CASE and the first CASE are not executed.

```
DO WHILE <exp>
    <statements>
ENDDO
```

Executes <statements> as long as <exp> evaluates as a logical TRUE. When <exp> evaluates to a logical FALSE, control is transferred to the statement following the ENDDO.

EDIT <n>

Displays the indicated record and allows the user to selectively change the contents of the data fields.

EJECT

Forces a form feed if the printer is turned on and either the SET PRINT ON or SET FORMAT TO PRINT command has been issued.

ELSE

Executes an alternate instruction path in an IF statement.

ENDCASE

Terminates a DO CASE statement.

ENDDO

Terminates a DO WHILE statement.

ENDIF

Terminates an IF statement.

ERASE

Clears the screen and places the cursor in the upper left corner of the screen.

FIND <char string>
FIND "<char string>"

Positions the current record pointer on the first record in an indexed data base whose key is the same as the character string. The data base file must be indexed and the index must be in USE.

GOTO [RECORD] <n>
GOTO TOP
GOTO BOTTOM
<n>
GOTO <memvar>

Repositions the record pointer of the file that is in USE. GO and GOTO may be used interchangeably. TOP positions the pointer on the first record in the data base, BOTTOM positions it on the last record. The other three forms of the command position the pointer on the indicated record number.

IF <exp>
  <statements>
[ELSE
  <statements>]
ENDIF

Allows conditional execution of commands. When <exp> evaluates to TRUE, the commands following the IF are executed. When <exp> evaluates to FALSE, the commands following the ELSE are executed. If no ELSE is specified, all commands to the ENDIF are skipped. IF commands may be nested to any level.

INDEX ON <exp> TO <index file>

Creates an <index file> for the file in USE. <exp> is the key to the index file; records are sequenced on that value.

INPUT ["<char string>"] TO <memvar>

Stores keyboard input to a character string memory variable. The memory variable is created if it does not already exist. The data type of <memvar> is determined from the type of data that is entered. If "<char string>" is present, it is displayed on the screen, followed by a colon as a prompt message. If it is not present, a colon is displayed on the screen as the prompt.

INSERT [BEFORE] [BLANK]

Adds one record to the data base file that is in USE. If BEFORE is present, the record is inserted before the current record, otherwise the new record is placed just after the current record. If BLANK is specified, a space-filled record is inserted, otherwise the user is prompted for input values in the data entry mode.

JOIN TO <file> FOR <exp> [FIELD <field list>]

Combines data from two data base files into a third file when some criterion is met. The two files used are the primary and secondary USE files. The current record pointer must be positioned on the first record of the primary USE file. JOIN evaluates the FOR <exp> for each record in the secondary USE file. Each time <exp> yields a logical TRUE, a record is added to the new file. When the end of the secondary USE file is reached, the primary USE file record pointer is advanced one record, the secondary USE file is repositioned on the first record, and the process continues until the primary USE file is exhausted.

If the FIELD phrase is omitted, the output file will consist of all the fields in the primary USE file's structure and as many of the secondary USE file's fields as will fit before exceeding the 32 field limit.

LIST [<scope>] [FOR <exp>] [<field list>] [OFF]

Lists all or part of the data base file in USE. If <scope> and FOR are not present, only the current record can contribute information for the LIST. If a FOR <exp> is present, the default becomes ALL. OFF inhibits listing the record number.

LIST FILES [ON <disk drive>] [LIKE *.<type>]

Lists names of files residing on a diskette. If there are no modifiers, DBF files are listed. ON allows the specification of a drive other than the default drive. Use the letter only to identify the drive; do not include the colon. The LIKE phrase allows other types of files to be listed. The <type> is the three character extension such as TXT, FRM, MEM, and so on.

LIST MEMORY

Lists all currently defined memory variables. Gives name, type, length, and contents for each variable and total number of bytes for all memory variables.

LIST STRUCTURE

Lists the structure of the USE data base. Lists name, type, length for each field, total record length, and usage such as date of last update and primary/secondary area assignment.

LOCATE [<scope>] [FOR <exp>]

Searches the records in the USE file for the first one for which <exp> is logically TRUE. When <exp> is satisfied, the message "RECORD n" is displayed. The CONTINUE command continues the search. Other dBASE II commands may be issued between LOCATE and CONTINUE.

LOOP

Redirects the flow of control in a DO WHILE command back to the DO WHILE clause.

MODIFY COMMAND [<command file>]

Creates a command file if the named file does not already exist. If the file exists, this command reads it from disk. Enters full-screen edit mode and allows additions, deletions, and changes to text in the file. Limits physical text lines to 77 characters. Interprets TAB characters as single spaces.

MODIFY STRUCTURE

Enters full-screen edit mode and allows changes to the structure of a data base file. Fields can be added and deleted, or the characteristics (type and length) can be changed. MODIFY STRUCTURE operates on the file that is in USE. It deletes ALL data records that are currently in the file.

NOTE [<char string>]

Places comments into a command file. <char string> is not displayed when the file is executed.

OTHERWISE

Executes an alternate path of instructions in a CASE structure.

PACK

Purges all records marked for deletion from the file that is in USE. If the file is indexed and the index file is in use, PACK adjusts the index file at the same time it adjusts the USE file. PACK does not free the unused space on the diskette.

POKE <address>, <byte list>

Stores a list of byte values into memory starting at the specified address.

QUIT [TO <CMD file list>]

Closes all data base files, command files, and alternate files and returns control to the operating system. If the TO phrase is present, all programs in <CMD file list> are executed in sequence by CP/M. There is no limit to the number of programs or CP/M commands which can be executed as long as the 254 character limit for dBASE II commands is not broken.

READ

Accepts data into GET commands in full-screen edit mode.

RECALL [<scope>] [FOR <exp>]

Removes deletion marks from records in the data base that is in use. The default for <scope> is the current record. If the FOR phrase is used, only those records that meet the criterion in <exp> are recalled.

RELEASE [<memvar list>]
[ALL]

Releases all or selected memory variables and makes the space they occupied available for new memory variables.

REMARK <char string>

Allows comments in a command file. Displays the comments on the output device when the file is executed.

RENAME <original file> TO <new file>

Changes the name of a file in the CP/M directory. If no file type is given, dBASE II assumes that the type is "DBF" and assigns that extension to the new file.

REPLACE [<scope>] <field> WITH <data> [,<field2> WITH <data2>...]
[FOR <exp>]

Replaces the contents of specified data field(s) of the file in USE with new data as specified in the corresponding data symbols. The <data> can be a constant, variable, or expression. The default for <scope> is the current record.

REPORT [FORM <form file>] [<scope>] [FOR <exp>] [TO PRINT]
[PLAIN]

Creates and/or prints reports by displaying data from the file in USE in a defined manner. The first time REPORT is used for a particular report, a FORM file (with the extension "FRM") is built. dBASE II prompts the user for specification of the report format. Once the form file is created, the report is automatically generated on the screen and/or the printer. The FORM phrase identifies an existing form file. <scope> defaults to ALL. The FOR phrase may be used to qualify the records to be included in the report. TO PRINT sends the report to the printer as well as the display screen. PLAIN suppresses printing the date and page number and is useful for producing tabular data.

RESET

Resets the CP/M bit map after a disk swap.

RESTORE FROM <mem file>

Reads the contents of a file of memory variables back into the computer's memory. The file must have been created with the SAVE command. Deletes any existing memory variables.

RETURN

Terminates a command file. Returns control to the command file which called the current command file, or to the keyboard if the command file was called from there.

SAVE TO <mem file>

Stores all currently defined memory variables to the named file with the entension "MEM".

SELECT [PRIMARY]
[SECONDARY]

Activates the named area for file access commands.

SET <parm> [ON]
[OFF]
SET <parm> TO <option>

Dynamically reconfigures the dBASE II environment. The <parm> field is a keyword, as described in Chapter 16. The value of <option> depends on the parameter and is also described in Chapter 16.

SKIP [+] [<var/literal/numeric exp>]
[-]

Advances or backs up the current record pointer relative to its present location.

SORT ON <field> TO <file> [ASCENDING]
[DESCENDING]

Sorts the file in USE to another file on the named field. The file that is in USE remains in USE and is unaltered. Records are sorted into ascending order unless otherwise specified.

STORE <exp> TO <memvar>

Saves the value of <exp> into the named memory variable, creating the variable if it does not already exist.

SUM <field list> [TO <memvar list>] [<scope>] [FOR <exp>]

Adds numeric expressions in the file in USE according to the <scope> and FOR phrases. Up to five fields can be summed. If the TO phrase is present, the sums are also stored into memory variables. The memory variables are created if they did not exist prior to the SUM command. The default for scope is all non-deleted records.

TOTAL ON <key> TO <file> [FIELD <field list>] [FOR <exp>]

Adds all or selected numeric fields in the file in USE and stores the totals in the named file. If the TO file does not exist, its structure is derived from the fields in the FIELD phrase. If there is no FIELD phrase, the structure for the file in USE is copied to the TO file.

UPDATE FROM <file> ON <key> [ADD <field list>]
[REPLACE <field list>]

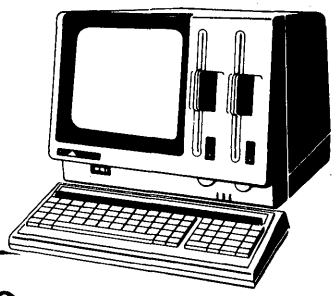
Revises the file in USE with data from another data base file. Both files must be ordered on <key>. The USE file can be sorted or indexed; the FROM file must be sorted. Updated fields can be summed or replaced in their entirety.

USE [<file>]
USE <file> INDEX <index file list>

Closes all open files in the active area and opens the named file, if any. When the INDEX option is used, multiple index files can be opened for the data base file that is named.

WAIT [TO <memvar>]

Pauses in program operation for keyboard input.



Appendix D

dBASE II Commands by Type of Operation

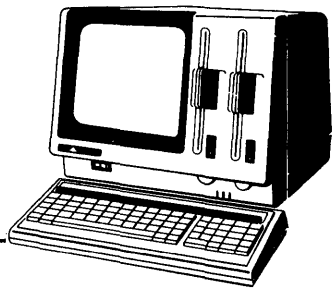
OPERATION	COMMAND
Create File	COPY CREATE INDEX JOIN MODIFY COMMAND REPORT SAVE TOTAL
Add Data To File	APPEND CREATE INSERT
Edit Data	BROWSE CHANGE DELETE EDIT READ RECALL REPLACE UPDATE

OPERATION	COMMAND
Display Data	? ?? @ BROWSE COUNT DISPLAY LIST READ REPORT SUM
Reposition Current Record Pointer	CONTINUE FIND GOTO LOCATE SKIP
Manipulate File	APPEND COPY DELETE DELETE FILE DO MODIFY STRUCTURE PACK RENAME SELECT SORT USE

OPERATION	COMMAND
Use Memory Variables	ACCEPT COUNT DISPLAY MEMORY INPUT LIST MEMORY RELEASE RESTORE SAVE STORE SUM WAIT

Program Command Files	ACCEPT
	CANCEL
	DO
	DO CASE
	IF
	ELSE
	ENDCASE
	ENDDO
	ENDIF
	INPUT
	LOOP
	MODIFY COMMAND
	OTHERWISE
	RETURN
	SET
	WAIT

Control The APC Environment	EJECT ERASE SET
Access the APC's Memory	POKE SET CALL TO CALL

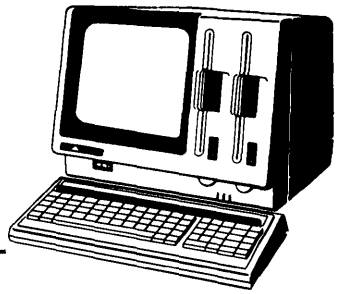


Appendix E

dBASE II Functions

FORMAT	TYPE	FUNCTION
!(<char string>)	C	Convert lowercase to uppercase.
#	N	Return the current record number.
$(<char string>,<start>,<length>)	C	Form a character string from part of another string.
&<mem var>	C	Replace the function symbol in the current command with the contents on the named memory variable.
*	L	Indicate whether current record is marked for deletion.
@(<char string1>,<char string2>)	N	Return starting point of character string1 relative to start of character string2.
CHR(<numeric exp>)	C	Return the ASCII character equivalent of the expression.
DATE()	C	Generate the system date.
EOF	L	Indicate whether the end of file has been reached.

FORMAT	TYPE	FUNCTION
FILE(<"file name"/var/exp>)	L	Indicate whether the file exists.
INT(<exp>)	N	Return the truncated integer value of the expression.
LEN(<var/string>)	N	Return the number of characters in the variable or string.
PEEK(<address>)	N	Return the value of the byte at the specified address.
STR(<numeric exp/var/number>, <length>, [<decimals>])	C	Convert numeric data to character string.
TEST(<exp>)	N	Check expression for syntactic correctness.
TRIM(<char string>)	C	Eliminate trailing blanks.
TYPE(<exp>)	C	Return character type of the data in the expression.
VAL(<char string>)	N	Convert character string to numeric data.



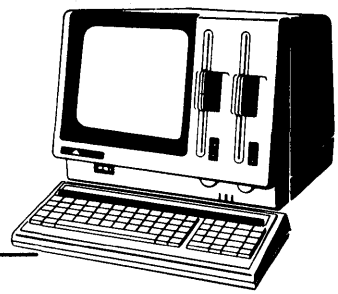
Appendix F

ASCII Character Codes

dBASE II uses the ASCII character codes listed in Table F-1. The table lists both the decimal value and the printed character. The collating sequence for characters is ascending order according to decimal values.

Table F-1 ASCII Character Codes

DECIMAL CODE	CHARACTER	DECIMAL CODE	CHARACTER	DECIMAL CODE	CHARACTER
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	¦
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		



Appendix G

dBASE II File Types

A file is a collection of information residing on a storage device such as a diskette. Data can be stored to, or retrieved from, a file. dBASE II files are grouped into seven types according to the kinds of data they contain, their organization, and how they are used by the system. All dBASE II files are standard CP/M files. File names conform to the following format:

[<disk drive>:]<file name>[.<extension>]

- <disk drive> is optional. If it is not specified, the default drive is assumed.
- <file name> is formed according to the same rules for all dBASE II file types: eight character limit, letters and numbers acceptable, must begin with a letter, no embedded spaces.
- <extension> identifies the file type. For each command that accesses a file, the extension may be omitted and dBASE II will assume the default type for that command. The defaults are shown in Table G-1.

Table G-1 File Type Extensions

FILE TYPE	EXTENSION
Data Base File	DBF
Memory File	MEM
Command (Program) File	PRG
Report Form File	FRM
Text File	TXT
Index File	NDX
Format File	FMT

Any legitimate CP/M file name may be used to refer to dBASE II files. Remember, if the file type is not supplied in a command, dBASE II assumes the default listed above. You may list files on a diskette by type with the command "LIST FILES

[LIKE *.<extension>]" where <extension> is the three-character extension. If the LIKE phrase is omitted, DBF file names are displayed.

DATA BASE FILES (DBF)

These files consist of one structure record and from zero to 65,535 data records. The extension is automatically assigned when a new file is CREATEd. These files should be modified *only* by dBASE II commands. Do not use a word processor or text editor to alter them.

REPORT FORM FILES (FRM)

Report formats are stored in files with the FRM extension. They are automatically created by dBASE II the first time a report is assigned a name with the REPORT command. They contain instructions to the report generator on headings, total fields, and column content identifiers. Report Form files can be modified using a word processor or text editor, however it is usually easier to define a new report from the scratch than to change an existing one.

COMMAND FILES (PRG)

The PRG files contain sequences of dBASE II command statements. They provide a way to save sets of frequently used command sequences and dBASE II programs. Command files may be created and modified with the MODIFY COMMAND instruction or with a word processor or text editor.

Command files may be nested, meaning that command files may contain DO commands that call other command files. Care should be taken when nesting command files so as not to exceed dBASE II's limitations. dBASE II allows up to 16 files to be open at any given time. Therefore, if there is a file in USE, only 15 command files may be nested. Certain commands also use work files. SORT uses two additional files; REPORT, INSERT, COPY, SAVE, RESTORE, and PACK use one additional file each. This further limits the number of available files. For instance, if a SORT command is issued from the lowest command file in a nest, only 13 levels of command file are left to be used (the USE file, 2 SORT work files, and 13 command files). When a command file issues the RETURN command or end-of-file is encountered in a command file, the command file is closed and its resources are again available for other commands.

INDEX FILES (NDX)

Index files consist of key values and pointers to data base file records. Index files are automatically created by the INDEX command. Indexing is a dBASE II technique that allows rapid location of data in a large data base.

MEMORY FILES (MEM)

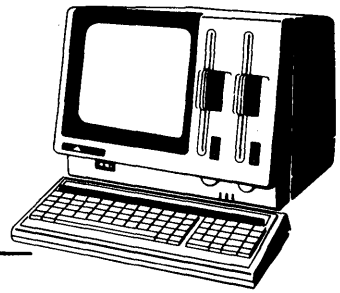
Memory files are static files of memory variables. The values of memory variables are independent of the data base in USE. Memory variables are used to contain constants, results of computations, and symbolic substitution strings. The SAVE command writes all current memory variables to a memory file. The RESTORE command reads a memory file back into the memory variables. In this way, memory variables can be reused from one dBASE II session to another. The maximum file size is 64 items, each up to 254 characters long.

TEXT FILES (TXT)

Text files are created in standard ASCII format by dBASE II under two conditions. Files created on disk using SET ALTERNATE TO <text file name> and SET ALTERNATE ON echo all screen output to the named file for session documentation. TXT files are also created by the COPY command with either the SDF or DELIMITED option.

FORMAT FILES (FMT)

Format files contain only @ and* commands. They are identified by the SET FORMAT TO <format file name> command and are activated by subsequent READ commands. Like command files, format files are created and modified by the MODIFY COMMAND instruction, a word processor, or a text editor. Format files are useful but not necessary. The commands they contain may be built into the command file that needs them.



Appendix H

A Sample Application

The sample application presented in this appendix uses a personnel data base to demonstrate programming techniques using dBASE II. Two files, EMPLOYEE.DBF and JOBDET.DBF, form the data base. They are developed in this guide. The version of EMPLOYEE.DBF that is distributed with dBASE II does not operate properly with the sample programs. It must be modified as described in this manual. You must also create JOBDET.DBF as described in this guide to run the sample application.

The programs that comprise the sample application are included on the dBASE II diskette you purchased. This appendix contains listings of the programs.

The sample application consists of ten separate programs, or procedures, that allow you to accomplish two major functions: updating the data base and reporting the contents of the data base.

1. DEMO.PRG

   This is the master menu program. It calls the UPDATE or REPORT procedure, or returns the system user to dBASE II or CP/M.

2. UPDATEDM.PRG

   This procedure directs the processing of file updates. Either the EMPLOYEE or JOBDET file can be updated. Records can be added, changed, deleted/recalled, or viewed.

3. REPORTDM.PRG

   This procedure directs report production. The system produces three reports that can be either displayed on the screen or printed.

4. VIEWDM.PRG

   This procedure is called by UPDATEDM.PRG. It allows the user to view all or part of the data base.

5. ADDM.PRG

   This procedure is called by UPDATEDM.PRG. It allows the user to add records to either file in the data base.

6. CHANGEDM.PRG

   This procedure is called by UPDATEDM.PRG. It allows the user to edit records, one field at a time.

7. DELETEDM.PRG

   This procedure is called by UPDATEDM.PRG. It allows the user to delete and recall records in the selected file.

8. REPT1DM.PRG

   This procedure displays or prints the average salary for each job.

9. REPT2DM.PRG

   This procedure prints or displays information about employees earning more than the midpoint of their salary range. This program uses the report form file MIDFORM.FRM.

10. REPT3DM.PRG

    This program prints or displays a salary increase report, using the parameter contained in the report form file SALINCR.FRM.

```
* DEMO: This is the system's master program. It calls command files UPDATEDM
* and REPORTDM, and it exits to either the dBASE prompt or CP/M (depending
* on user input).

* display colons at boundaries of 'get' fields
set colon on
set screen on
* do not display dBASE messages
set talk off
* program cannot be halted with <ESC>
set escape off
set bell off
* saves current memory variables
save to tempdm
set print off
set format to screen
* create and initialize memory variables
store '                    ' to spaces
store '  ' to menu:let
store 0 to menu:num
do while t
  erase
  @ 5,30 say 'MASTER MENU'
  @ 8,26 say '1) Update Database'
  @ 10,26 say '2) Report'
  @ 12,26 say '3) Return to DBASE prompt'
  @ 14,26 say '4) Return to CP/M'
  @ 18,18 say 'Please choose one ' get menu:num picture '9'
  read
  do case
    case menu:num = 1
      do updatedm
    case menu:num = 2
      do reportdm
    case menu:num = 3
      select primary
      use
      set talk on
      set bell on
      restore from tempdm
      return
```

```
        case menu:num = 4
           quit
      endcase
   enddo t
```

```
* UPDATEDM: This program directs processing of updates. The user selects a
* database, which is loaded into the primary area. Then a second menu offers
* a number of update options.
do while t
   erase
   @ 2,30 say 'UPDATE MENU'
   @ 5,25 say '1) Update Employee Database'
   @ 7,25 say '2) Update Job Desc. Database'
   @ 9,25 say '3) Return to MASTER MENU'
   @ 11,18 say 'Please choose one ' get menu:num picture '9'
   read
   do case
      case menu:num = 3
         return
* This block erases unchosen menu options from the display and loads
* the appropriate database into the primary area.
case menu:num = 1 .or. menu:num = 2
   @ 9,18 say spaces
   @ 11,18 say spaces
   select primary
   if menu:num = 1
      @ 5,25 say '* '
      @ 7,25 say spaces
      ?
      use employee
   else
      @ 5,25 say spaces
      @ 7,25 say '* '
      ?
      use jobdet
   endif 1
   store t to update:on
   * Update loop. Store f to update:on if another database is to be
   * loaded. (Returns to first menu.)
   do while update:on
      store 0 to menu:num2
      @ 10,25 say '1) View Database'
      @ 12,25 say '2) Add Records'
      @ 14,25 say '3) Change Records'
      @ 16,25 say '4) Delete/Recall Records'
      @ 18,25 say '5) Update Another Database'
```

```
            @ 20,25 say '6) Return to MASTER MENU'
            @ 23,18 say 'Please choose one ' get menu:num2 picture '9'
            read
            do case
               case menu:num2 = 1
                  do viewdm
               case menu:num2 = 2
                  do addm
               case menu:num2 = 3
                  do changedm
               case menu:num2 = 4
                  do deletedm
               case menu:num2 = 5
                  store f to update:on
               case menu:num2 = 6
                  return
            endcase
            * Menu:num2 = 0 when user chooses to return to MASTER MENU from
            * VIEWDM, ADDM, CHANGEDM, or DELETEDM.
            if menu:num2 = 0
               return
            endif 0
         enddo update:on
      endcase
   enddo t
```

```
* REPORTDM: This program gives user 3 report options: REPT1DM, REPT2DM,
* and REPT3DM (q.v.). Control returns to DEMO only if menu option #4 is
* chosen. Reports may be displayed or printed.
store 0 to menu:num
do while t
  erase
  @ 5,30 say 'REPORT MENU'
  @ 8,10 say '1) Average salary for each job title'
  @ 10,10 say '2) Employees earning more than midpoint of salary range'
  @ 12,10 say '3) Salary increase (by Department)'
  @ 14,10 say '4) Return to MASTER MENU'
  @ 17,5 say 'Please choose one ' get menu:num picture '9'
  read
  if menu:num >0 .and. menu:num <4
    store t to true2
    do while true2
      @ 20,15 say 'Do you want a printed copy of this report? (y,n)';
        get menu:let picture 'A'
      read
      if menu:let = 'y'
        set format to print
        set print on
        store f to true2
      else
        if menu:let = 'n'
          @ 22,20 say ;
            '(Press <CTRL> S to stop scrolling, any key to continue.)'
          set console off
          wait
          set console on
          store f to true2
        endif n
      endif y
    enddo true2
  endif 0,4
  do case
    case menu:num = 1
      do reptldm
    case menu:num = 2
      do rept2dm
```

```
      case menu:num = 3
         do rept3dm
      case menu:num = 4
         return
   endcase
   if menu:num >= 1 .and. menu:num <= 3
      set print off
      set format to screen
      ?
      ? 'Press any key to continue'
      set console off
      wait
      set console on
   endif
enddo t
```

```
* VIEWDM: This program allows users to view part or all of the database
* chosen in UPDATEDM.
select primary
count to reccount
copy structure extended to structx
select secondary
use structx
do while t
   erase
   store t to true2
* input record #'s
do while true2
   store 0 to startrec
   @ 15,5 say 'List from record #' get startrec picture '99999'
   read
   if startrec > 0 .and. startrec <= reccount
      select primary
      go startrec
      * main loop
      do while .not. eof
         erase
         store 1 to accum
         @ accum, 1 say 'Record #' + str (#,5)
         if *
            @ 1,30 say 'DELETED'
         endif *
         select secondary
         go top
         * print field names and contents
         do while .not. eof
            store accum + 1 to accum
            store field:name to field
            @ accum, 1 say field:name + ': '
            @ accum,len(field:name) + 3 say &field
            skip
         enddo .not. eof
         select primary
         @ 23,10 say "Press any key to continue ('q' to quit)"
         set console off
         wait to waitvar
```

```
            set console on
            if !(waitvar) = 'Q'
               go bottom
            endif !
            skip
         enddo .not. eof
         store f to true2
      endif > 0 .and. <= reccount
   enddo true2
   erase
   store t to true2
   do while true2
      @ 5,30 say 'VIEW MENU'
      @ 8,25 say '1) List records'
      @ 10,25 say '2) Return to UPDATE MENU'
      @ 12,25 say '3) Return to MASTER MENU'
      @ 15,25 say 'Please choose one ' get menu:num2 picture '9'
      read
      do case
         case menu:num2 = 1
            store f to true2
         case menu:num2 = 2
            erase
            return
         case menu:num2 = 3
            store 0 to menu:num2
            return
      endcase
   enddo true2
enddo t
```

```
* ADDM: This program adds records to the database chosen in UPDATEDM.
select primary
copy structure extended to structx
select secondary
use structx
do while t
  set colon off
  erase
  * line counter
  store 1 to accum
  select primary
  append blank
  @ accum,1 say 'Record #' + str(#,5)
  select seco
  go top
  * display field names
  do while .not. eof
    store accum + 1 to accum
    @ accum, 1 say field:name + ': '
    skip
  enddo .not. eof
  go top
  store 1 to accum
  * loop until all fields have been entered
  do while .not. eof
    store field:name to field
    store accum + 1 to accum
    do case
      * get character field
      cate field:type = 'C'
      store t to true2
        do while true2
          store spaces to charin
          @ accum,len(field:name) + 3 get charin ;
            picture 'XXXXXXXXXXXXXXXXXXXX'
          read
          if len(trim(charin)) <= field:len
            store f to true2
            replace &field with charin
          endif <= field:len
        enddo true2
```

```
        * get logical field
        case field:type = 'L'
          store t to login
          @ accum,len(field:name) + 3 get login picture 'A'
          read
          replace &field with login
        * get numerical field
        otherwise
          store t to true2
          do while true2
            store 0 to numin
            @ accum,len(field:name) + 3 get numin picture '99999999'
            read
*           data check
              replace &field with numin
              store f to true2
*           *end check
          enddo true2
        endcase
        skip
      enddo .not. eof
      set colon on
      store t to true2
      do while true2
        erase
        @ 5,30 say 'ADD MENU'
        @ 8,25 say '1) Add another record'
        @ 10,25 say '2) Return to UPDATE MENU'
        @ 12,25 say '3) Return to MASTER MENU'
        @ 15,18 say 'Please choose one ' get menu:num2 picture '9'
        read
        do case
          case menu:num2 = 1
            store f to true2
          case menu:num2 = 2
            erase
            return
          case menu:num2 = 3
            store 0 to menu:num2
            return
```

```
            endcase
        enddo true2
    enddo t
```

```
* CHANGEDM: This program edits records, one field at a time.
select primary
* count # records (for input checking)
count to reccount
copy structure extended to structx
select secondary
use structx
do while true
  erase
  store t to true2
  * input record #'s
  do while true2
    store 0 to lorec, hirec
    @ 10,0 say 'Edit record #' get lorec picture '99999'
    @ 10,24 say 'thru record #' get hirec picture '99999'
    @ 11,37 say '<RETURN> if only one'
    @ 12,35 say 'record is to be changed'
    read
    if hirec = 0
      store lorec to hirec
    endif
    if hirec - lorec >= 0 .and. .not. (hirec > reccount) .and. lorec > 0
      store f to true2
    endif
  enddo true2
  set colon off
  * execute once/record
  do while lorec <= hirec
    erase
    select primary
    go lorec
    @ 1,2 say 'Record #' + str(#,5)
    if *
      @ 1,30 say 'DELETED'
    endif
    select secondary
    go top
    store 0 to fieldno
    * display field names and contents
    do while .not. eof
```

```
   store field:name to fieldname
   store fieldno + 1 to fieldno
   @ fieldno + 1,1 say str(fieldno,2) + ') ' + field:name + ': '
   @ fieldno + 1,len(field:name) + 7 say &fieldname
   skip
enddo .not. eof
@ fieldno + 2,1 say str(fieldno + 1,2) + ') ';
   + '**Proceed to next record'
store t to true3
* execute until changes to current record are complete
do while true3
   store t to true2
   * input field number
   do while true2
      store 0 to menu:num2
      @ fieldno + 4,2 say 'Please choose one (<RETURN> to quit) ' ;
         get menu:num2 picture '99'
      read
      * if <RETURN>, quit
      if menu:num2 = 0
         store hirec + 1 to lorec
         store f to true2, true3
      else
         * skip to next record
         if menu:num2 = fieldno + 1
            store lorec + 1 to lorec
            store f to true2, true3
         else
            if menu:num2 > 0 .and. menu:num2 <= fieldno
               store f to true2
            endif > 0
         endif = fieldno + 1
      endif = 0
   enddo true2
   * change block
   if menu:num2 0 > .and. menu:num2 <= fieldno
      go menu:num2
      store field:name to fieldname
      do case
         * change character field
         case field:type = 'C'
```

```
                store t to true2
                do while true2
                  store spaces to charin
                  @ menu:num2 + 1,len(field:name) + field:len + 10 ;
                    say ':: ' get charin picture 'XXXXXXXXXXXXXXXXXXX'
                  read
                  if len(trim(charin)) <= field:len
                    store f to true2
                    replace &fieldname with charin
                    @ menu:num2 + 1,len(field:name) + 7 say charin
                  endif <= field:len
                enddo true2
              * change logical field
              case field:type = 'L'
                store t to login
                @ menu:num2 + 1,len(field:name) + field:len + 10 ;
                  say ':: ' get login picture 'A'
                  read
                  replace &fieldname with login
                  @ menu:num2 + 1, len(field name) + 7 say login
               * change numerical field
               otherwise
                  store t to true2
                  do while true2
                    store 0 to numin
                    @ menu:num2 + 1,len(field:name) + field:len + 10 ;
                      say ':: ' get numin picture '99999999'
                    read
*                   if len(trim(str (numin,))) <= field:len
                      replace &fieldname with numin
                      @ menu:num2 + 1,len(field:name) + 7 say &fieldname
                      store f to true2
                    *endif <= field:len
                  enddo true2
            endcase
            * erase display beyond new field (in case of input beyond
            * allowable length)
            @ menu:num2 + 1,len(field:name) + field:len + 10 say spaces
          endif = fieldno
        enddo true3
      enddo lorec = hirec
```

```
    store t to true2
    do while true2
      set colon on
      erase
      @ 5,30 say 'CHANGE MENU'
      @ 8,25 say '1) Edit records'
      @ 10,25 say '2) Return to UPDATE MENU'
      @ 12,25 say '3) Return to MASTER MENU'
      @ 15,18 say 'Please choose one ' get menu:num2 picture '9'
      read
      erase
      do case
        case menu:num2 = 1
          store f to true2
        case menu:num2 = 2
          return
        case menu:num2 = 3
          store 0 to menu:num2
          return
      endcase
    enddo true2
enddo t
```

```
* DELETEDM: This program deletes and recalls records from database chosen
* in UPDATEDM.
erase
select primary
count to reccount
do while t
  @ 5,30 say 'DELETE MENU'
  @ 8,25 say '1) Delete a record'
  @ 10,25 say '2) Recall a record'
  @ 12,25 say '3) Return to UPDATE MENU'
  @ 14,25 say '4) Return to MASTER MENU'
  @ 17,18 say 'Please choose one ' get menu:num2 picture '9'
  read
  do case
    case menu:num2 = 1
      store t to true2
      * input loop
      do while true2
        store 0 to recnum
        @ 21,28 say 'Delete record #' get recnum picture '99999'
        read
        if recnum > 0 .and. recnum <= reccount
          go recnum
          * display DELETE message
          set talk on
          delete
          set talk off
          store f to true2
        endif >0 .and. <= reccount
      enddo true2
    case menu:num2 = 2
      store t to true2
      do while true2
        store 0 to recnum
        @ 21,28 say 'Recall record #' get recnum picture '99999'
        read
        if recnum > 0 .and. recnum <= reccount
          go recnum
          * display RECALL message
          set talk on
          recall
```

```
            set talk off
            store f to true2
          endif > 0 .and. <= reccount
        enddo true2
      case menu;num2 = 3
        erase
        return
      case menu:num2 = 4
        store 0 to menu:num2
        return
  endcase
enddo t
```

```
* REPT1DM: This program displays or prints average salary for each job
* title.
erase
@ 2,30 say 'AVERAGE SALARIES'
@ 4,10 say 'Job Title'
@ 4,35 say 'Job Code'
@ 4,50 say '# Emp.'
@ 4,62 say 'Average Salary'
* line counter
store 7 to accum
select primary
use employee
select secondary
use jobdet
* executes once/job title
do while .not. eof
   select primary
   sum salary for job:code = s.job:code to sumsal
   count for job:code = s.job:code to jobcount
   if jobcount = 0
      * to prevent zero divide
      store 0.0 to average
   else
      store sumsal/jobcount to average
   endif
   @ accum,10 say job:title
   @ accum,39 say s.job:code
   @ accum,43 say jobcount
   @ accum,63 say average
   select secondary
   skip
   store accum = 1 to accum
enddo
select primary
return
```

```
* REPT2DM: This program prints or displays information about employees
* earning more than the midpoint of their salary range.
erase
select secondary
use jobdet
select primary
use employee,
* create new file with fields from employee (last name, first name, salary)
* and jobdet (job title, low salary, high salary)
join to midpoint for job:code = s.job:code field first:name, last:name,;
   job:title, low:sal, hi:sal,salary
use midpoint
* second line of heading is within midform
set heading to   Employees Earning More Than
report form midform for salary > (low:sal + hi:sal)/2
set heading to
return
```

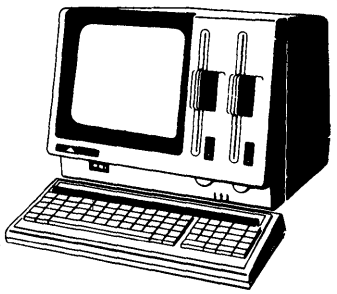
```
* REPT3DM: This program prints or displays a salary increase report.
* Amount of increase is supplied by user at runtime.
erase
store 0 to percent
@ 14,3 say 'Percentage Increase? (5, 12, etc.)' get percent picture '999'
read
@ 16,3
* get title
accept 'Report Title? ' to headng
erase
if len(trim(headng)) < 80
   * center heading if < 80 characters
   @ 4, (80- len(trim(headng)))/2 say headng
else
   @ 4,0 say headng
endif
use employee
index on dept:num to dptndxdm
report form salincr plain
return
```

SAMPLE APPLICATION REPORT FORM FILES

The sample application uses two report form files. REPT2DM.PRG calls MIDFORM.FRM, REPT3DM.PRG calls SALINCR.FRM. Table H-1 lists the report parameter values for these two form files.

Table H-1. Report Form File Parameters

REPORT PROMPT	MIDFORM.FRM	SALINCR.FRM
PAGE HEADING? (Y/N)	y	n
PAGE HEADING	Midpoint of Salary Range	
DOUBLE SPACE REPORT? (Y/N)	n	n
TOTALS REQUIRED? (Y/N)	n	y
SUBTOTALS REQUIRED? (Y/N)		y
SUBTOTALS FIELD		dept:num
SUMMARY REPORT? (Y/N)		n
EJECT PAGE AFTER SUBTOTALS?		n
SUBTOTAL HEADING		Department
001	20,TRIM(FIRST:NAME) +' '+LAST:NAME	20, TRIM(FIRST:NAME) +' '+LAST:NAME
HEADING	Employee	Employee
002	20,JOB:TITLE	8, SALARY
HEADING	Job Title	Current Salary
TOTALS REQUIRED? (Y/N)		y
003	9,LOW:SAL	28,' increased by ' + str (PERCENT,3) + '% --- '
HEADING	Minimum Salary	
004	9,HI:SAL	10, SALARY * (100 + PERCENT)/100
HEADING	Maximum Salary	New Salary
TOTALS REQUIRED? (Y/N)		y
005	9,SALARY	
HEADING	Current Salary	



Appendix I

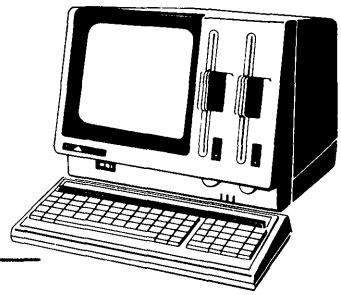
Function Keys

At the top of the APC keyboard are a key labelled FNC and 22 function keys. The most frequently used dBASE II commands are assigned to the first twelve function keys. A template is included with the dBASE II documentation package. This template, placed above the function keys on the keyboard, indicates the command assigned to each function key. Press the appropriate function key to execute these commands. When you use a function key, you do not have to type the command. When the function key is pressed, up to eight characters of the corresponding command appear on the display screen. If the command requires or accepts modifiers, type your entry immediately following the display.

Two word commands, such as MODIFY COMMAND, SELECT PRIMARY, and SELECT SECONDARY can be entered with two function keys.

Table I-1 Function Keys for dBASE II

FUNCTION KEY	COMMAND	SCREEN DISPLAY
1	USE	USE
2	LIST	LIST
3	DISPLAY	DISPLAY
4	CREATE	CREATE
5	APPEND	APPEND
6	EDIT	EDIT
7	SELECT	SELE
8	PRIMARY	PRIM
9	SECONDARY	SECO
10	MODIFY	MODI
11	COMMAND	COMM
12	ERASE	ERASE



Appendix J

Ideas and Applications

Ideas and Applications is intended to be a collection of short, useful dBASE II programs and techniques that may ease the process of data management for other dBASE II users. The programs are examples of how something may be done with dBASE, and it is hoped that other dBASE users can take these ideas and move beyond.

Ashton-Tate urges anyone with a short program in dBASE II that does something particularly difficult, handy, or unusual to send it along. It is understood that these programs are written by the contributor and are given gratis into the public domain. Credit will be given to the contributors by name and city. With a language as powerful as dBASE we know that there is a wealth of material out there. Discovery and fixes of bugs are also welcome.

Ashton-Tate will select and edit as necessary, and mail Ideas and Applications from time to time to registered owners of dBASE II. The programs can then be added to the back of this manual. Ashton-Tate is especially interested in applications of the BROWSE, PEEK, POKE, and CALL commands.

Ashton-Tate hopes that this is a valuable service, and would appreciate your written comments.

ASHTON-TATE
9929 W. JEFFERSON BLVD.
CULVER CITY, CA 90230

```
*                              II COMMAND

* II.CMD by                   BILL WEINMAN
*                             LOS ANGELES, CA

* additional thanks to        STEVE EDIGER

* THIS dBASE COMMAND FILE DISPLAYS A MENU OF USER OPTIONS.
* YOU CAN SET THE SYSTEM DATE, RETURN TO dBASE, DO THE RUN
* COMMAND, OR ANY OTHER FUNCTION YOU WOULD LIKE TO ADD.
* II.CMD COMBINED WITH RUN.CMD IS A CONVENIENT TURNKEY
* UTILITY FOR EXIT TO ANY CP/M FILE AND RETURN TO dBASE.
* THE OPTION TO SET THE SYSTEM DATE ALLOWS A SYSTEM DATE IN
* A FORMAT OTHER THAN "MM/DD/YY", AND THEREBY GETS
* AROUND THE INVALID DATE CHECK DONE BY dBASE. FOR EXAMPLE,
* USE A EUROPEAN DATE FORMAT "DD/MM/YY"; OR, DO AN ABSOLUTE
* DATE COMPARISON BY ASCII CHARACTER WITH A FORMAT LIKE
* "YY/MM/DD" (WITH < OR > ). BRING UP 'II.CMD' AUTOMATICALLY
* BY TYPING 'DBASE II' FROM CP/M.

SET TALK OFF
DELETE FILE $$$.SUB
SET BELL OFF
SET SCREEN ON
SET FORMAT TO SCREEN
SET ECHO OFF
SET CONSOLE ON
SET ALTERNATE OFF
```

```
* MENU SELECTION
* --------------
STORE T TO MAINFLG
DO WHILE MAINFLG
  ERASE
  ?
  ?
  ?
  ?
  ?
  ?"          THIS IS WHAT YOU MAY NOW DO:"
  ?
  ?
  ?"          1 - SET DATE"
  ?"          2 - dBASE II COMMAND SUMMARY"
  ?"          3 - RUN A CP/M PROGRAM"
  ?"          4 - RETURN TO dBASE"
  ?
  ?"          SELECT FROM ABOVE CHOICES"
  ?
  *

  II.CMD CONTINUED
  *
  STORE "1" TO Mchoice
  @ 19,5 SAY "ENTER YOUR SELECTION ->" GET Mchoice PICTURE "#"
  READ

  DO CASE
  CASE Mchoice = "1"
    SET CONFIRM ON
    SET COLON OFF
    ERASE
    STORE "          " TO Mdate
   @ 10,20 SAY "ENTER DATE (MM/DD/YY): " GET Mdate PICTURE '99/99/99'
    READ
    SET DATE TO &Mdate
    SET COLON ON
    SET CONFIRM ON
```

```
   CASE Mchoice = "2"
      DO CMDS

   CASE Mchoice = "3"
      DO RUN

   CASE Mchoice = "4"
      SET TALK ON
      SET BELL ON
      RETURN

   ENDCASE
ENDDO MAINFLG
SET TALK ON
RETURN
```

```
*                              RUN COMMAND

* RUN.CMD by                  BILL WEINMAN
*                             LOS ANGELES, CA

* additional thanks to        STEVE EDIGER

ERASE
REMA- THIS dBASE COMMAND FILE WILL EXIT dBASE TO AUTOMATI-
REMA- CALLY EXECUTE A DIR, TYPE, SAVE, ERA, OR ANY CP/M .COM
REMA- FILE THAT EXISTS ON THE CURRENTLY LOGGED DRIVE.
REMA- AFTER COMPLETION OF THE COMMAND, RUN RETURNS TO
REMA- dBASE. IN THIS EXAMPLE, RUN STARTS EXECUTION OF
REMA- ' II.CMD', WHICH IS ATTACHED.

SET TALK OFF
STORE "                    " TO Mcom
DO WHILE T
  @ 10,20 SAY "ENTER COMMAND -> " GET MCom PICTURE "!!!!!!!!!!!!!!!!!!!!"
  READ
  STORE ! ($ (Mcom, 1, @ (" ",Mcom)-1)) TO Verb

  IF Verb "TYPE";
    .OR. Verb = "DIR";
    .OR. Verb = "SAVE";
    .OR. Verb = "ERA";
    .OR. FILE (Verb+".COM")

      DO CASE
      CASE Verb = "TYPE"
        @ 12,22
        @ 12,22 SAY "TYPE ctrl-S TO STOP/START SCROLLING."
        WAIT
      CASE Verb = "DIR"
        @ 12,22
        @ 12,22 SAY "TYPE ctrl-S TO STOP/START SCROLLING."
        WAIT
      ENDCASE
      ERASE

      STORE TRIM(Mcom) TO TMcom
      QUIT TO "&TMcom","DBASE II"
```

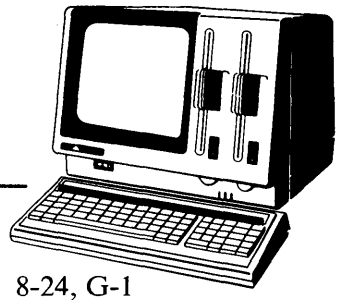
```
   ELSE
        @ 12,20
        @ 12,20 SAY "FILE "+Verb+".COM DOES NOT EXIST."

   ENDIF FILE
 ENDDO T

*********************************************************************
* REMA- THIS PROGRAM WILL CALCULATE THE SQUARE ROOT OF A*
* REMA- NUMBER                                                      *
*********************************************************************
* CONTRIBUTED BY NELSON TSO                                         *
*********************************************************************
*
STORE 1.00000000 TO ROOT
STORE "          " TO NUMBER
SET TALK OFF
ERASE
STORE T TO ENTER
DO WHILE ENTER
@ 3,3 SAY "ENTER NUMBER TO BE ROOTED " GET NUMBER PICTURE '9999999999'
READ
@ 5,3
IF "-" $NUMBER
   @ 5,3 SAY "NO '-' IN THE ENTRY PLEASE"
   LOOP
ENDIF
IF &NUMBER > 99.9999999
   @ 5,3 SAY "NUMBER IS TOO LARGE"
   LOOP
ENDIF
IF &NUMBER < 0
   @ 5,3 SAY "ERROR IN ENTRY"
   LOOP
ENDIF
STORE F TO ENTER
ENDDO ENTER
IF &NUMBER = 0
   @ 5,3 SAY "ROOT = 0"
   STORE F TO B
```

```
ELSE
  STORE T TO B
ENDIF
DO WHILE B
  STORE &NUMBER/(ROOT*ROOT) - 1 TO C
  IF C <0
    STORE C*(-1) TO C
  ENDIF
  IF C < 0.0000001
    @ 5,3 SAY "ROOT = "
    @ 5,15 SAY ROOT
    STORE F TO B
  ELSE
    STORE (&NUMBER/ROOT + ROOT)/2 TO ROOT
    STORE T TO B
  ENDIF
ENDDO
RETURN
NOTE- SQUARE ROOTS OF NUMBERS LARGER THAN 99 CAN BE
NOTE- CALCULATED BY DIVIDING THE NUMBER BY 10000 OR 100
NOTE- AND MULTIPLYING THE RESULTING SQUARE ROOT BY 100 OR
NOTE- 10 RESPECTIVELY.
```

```
*                              SEARCH AND REPLACE
* SEARCH.CMD by                  JIM TAYLOR
*                                ASHTON-TATE
*
* THIS PROGRAM IS AN EXAMPLE OF HOW TO LOCATE PARTICULAR
* CHARACTER SUBSTRINGS AND REPLACE THEM TO ACHIEVE
* UNIFORMITY IN THE ENTIRE DATABASE FILE. THIS IS ESPECIALLY
* USEFUL IF UNWANTED MARKERS CAME ALONG WITH YOUR
* DATA WHEN BROUGHT INTO dBASE FROM A FOREIGN FILE
* STRUCTURE. EVEN INVISIBLE CHARACTERS CAN BE DELETED. BE
* CAREFUL TO KEEP THE SUBSTRING LENGTHS CORRECT.

ERASE
@ 8,10 SAY "                SEARCH "
@ 10,10 SAY "THIS PROGRAM REMOVES ALL ' von ',' vom ',' v. ',' der ', etc. "
@ 12,10 SAY "        AND REPLACES THEM WITH ' v ', AND ' d ' "
SET TALK ON
SET ECHO ON
ACCEPT "REMOVE 'VONS' FROM WHICH FILE? " TO FILE
USE &FILE
* WORKS ON FIELD CALLED 'NAME' WHICH IS 50 CHARACTERS LONG
DO WHILE .NOT. EOF
  IF " von "$name
    STORE @(" von ", name) TO V
    REPLACE name WITH $(name,1,V)+"v "+$ (name,V+5,45-V)
  ELSE
    IF " v. "$name
      STORE @(" v. ",name) TO V
      REPLACE name WITH $ (name,l,V)+"v "+S (name,V+4,46-V)
    ELSE
      IF " vom "$name
        STORE @(" vom ",name) TO V
        REPLACE name WITH $(name,1,v)+"v "+$(name,V+4,46-V)
      ELSE
        IF " van "$name
          STORE @(" van ",name) TO V
          REPLACE name WITH $(name,1,v)+"v "+$(name,V+5,45-V)
        ENDIF
      ENDIF
    ENDIF
  ENDIF
```

```
    *
    IF " d. "$name
      STORE @(" d. ",name) TO V
      REPLACE name WITH $ (name, 1,V)+"d "+$ (name,V+4,46-V)
    ELSE
      IF " der "$name
        STORE @(" der ",name) TO V
        REPLACE name WITH $ (name, 1,V)+"d "+$ (name,v+5,45-V)
      ENDIF
    ENDIF
  RELEASE ALL
  SKIP
  ENDDO
  * END OF SEARCH.CMD
```

Index



A

D

T

U

Advanced Personal Computer
TM

NEC
NEC Information Systems, Inc.

USER'S COMMENTS FORM

Document: dBase II User's Guide

Document No.: 819-000100-8001

Please suggest improvements to this manual.

Please list any errors in this manual. Specify by page.

From:

Name _____

Title _____

Company _____

Address _____

Dealer Name _____

Date: _____

Please cut along this line.

Seal or tape all edges for mailing-do not use staples.

FOLD HERE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 386 LEXINGTON, MA

POSTAGE WILL BE PAID BY ADDRESSEE

NEC Information Systems, Inc.
Dept: Publications –APC
5 Militia Drive
Lexington, MA 02173

FOLD HERE

Seal or tape all edges for mailing-do not use staples.